AAA_LOGIN_T

Operates on AAA login request information

This topic lists the expressions that are provided by this class.

DOMAIN

Returns the domain of the user. This is the domain

used at the time of login

Returns: text_t

PASSWORD

Returns the password of the user. This is the password

used at the time of login

Returns: text_t

PASSWORD2

Returns the password2 of the user. This is the password2

used at the time of login

Returns: text_t

USERNAME

Returns the name of the user. This is the name used

at the time of login

Returns: text_t

VALUE(text_tname)

Returns the value associated with the parameter name. The

value is extracted at the time of login from the name-value

pair within the POST body

Parameters(expressions not allowed):

name -

```
the name of the name-value pair whose value is
```

to be extracted

Returns: text_t

AAA_AUTHENTICATION_T

Operates on AAA authentication information.

This topic lists the expressions that are provided by this class.

ERROR

Returns error encountered in current authentication factor.

Returns: authentication_error_e

NONCE

Returns nonce for current authentication factor.

Returns: text_t

AAA_T

Operates on AAA information.

This topic lists the expressions that are provided by this class.

AUTHENTICATION

Returns AAA authentication information.

Returns: aaa_authentication_t

LOGIN

Returns the AAA login request information.

Returns: aaa_login_t

USER

Returns the AAA User associated with the current transaction.

Returns: aaa_user_t

AAA_USER_T

Represents the AAA User Information.

This topic lists the expressions that are provided by this class.

ATTRIBUTE(uint index)

Returns USER's attribute stored at a given index.

For example:

HTTP.REQ.USER.ATTRIBUTE(1) gives the value of a previously

extracted attribute that is stored as 'attribute1'.

Index range is between 1 and 16, both included.

Parameters:

index -

```
offset
```

Returns: text_t

ATTRIBUTE(text_tname)

Returns USER's attribute stored with a given name.

For example:

HTTP.REQ.USER.ATTRIBUTE("name") gives the value of a previously

extracted attribute that is stored as 'attribute'.

Parameters:

name -

```
name used for extracting corresponding attribute value
```

Returns: text_t

AUTH_STRENGTH

Returns an integer that represents the strength of

authentication of the current user session.

Returns: num_at

DOMAIN

Returns the Domain of the user.

Returns: text_t

EXTERNAL_GROUPS

Operate as a List on the External Groups which is seperated by ','.

For example:

HTTP.REQ.USER.EXTERNAL_GROUPS gives list of all groups which are

',' seperated.

Returns: list_t

EXTERNAL_GROUPS(char sep)

Operate as a List on the Group which is seperated by given

delimiter.

For example:

HTTP.REQ.USER.EXTERNAL_GROUPS(':') gives list of all external

groups which are ':' seperated.

Parameters(expressions not allowed):

sep -

```
delimiter
```

Returns: list_t

GROUPS

Operate as a List on the Group which is seperated by ','.

For example:

HTTP.REQ.USER.GROUPS gives list of all groups which are

',' seperated. In this list first internal groups are listed

which are followed by external groups.

Returns: list_t

GROUPS_AS_XML(text_ttag)

Gives the list of groups in a SAML assertion multi-attribute format,

each enclosed by an element that has the specified tagname.

For example:

HTTP.REQ.USER.GROUPS_AS_XML("mytagname"): the expression will return

a list of groups in the following format -

financeenggIn this example, finance, engg are the group names.

Parameters:

tag -

```
tagname
```

Returns: text_t

GROUPS(char sep)

Operate as a List on the Group which is seperated by given

delimiter.

For example:

HTTP.REQ.USER.GROUPS(':') gives list of all groups which are

':' seperated. In this list first internal groups are listed

which are followed by external groups.

Parameters(expressions not allowed):

sep -

```
delimiter
```

Returns: list_t

INTERNAL_GROUPS

Operate as a List on the Group which is seperated by ','.

For example:

HTTP.REQ.USER.INTERNAL_GROUPS gives list of all groups which are

',' seperated.

Returns: list_t

INTERNAL_GROUPS(char sep)

Operate as a List on the Group which is seperated by given

delimiter.

For example:

HTTP.REQ.USER.INTERNAL_GROUPS(':') gives list of all internal groups

which are ':' seperated.

Parameters(expressions not allowed):

sep -

```
delimiter
```

Returns: list_t

IS_MEMBER_OF_ALL(text_tpatset_name)

Returns TRUE if all patterns(group string) of the patset

matches user's groups in session

Parameters(expressions not allowed):

patset_name -

```
patset Name
```

Returns: bool_at

IS_MEMBER_OF_ANY(text_tpatset_name)

Returns TRUE if there is at least one pattern(group string) of the patset

matches user's groups in session

Parameters(expressions not allowed):

patset_name -

```
patset Name
```

Returns: bool_at

IS_MEMBER_OF(text_tgroup_name)

Returns TRUE if the user is a member of the group group_name

Parameters(expressions not allowed):

group_name -

```
Group Name
```

Returns: bool_at

LOGIN_ATTEMPTS

Return the number of failed login attempts for the current user.

User name is extracted from current authentication process.

Returns: num_at

LOGIN_ATTEMPTS(text_tusername)

Return the number of failed login attempts for the given user.

Parameters(expressions not allowed):

username -

```
the name of the user to get the information for.
```

Returns: num_at

LOGIN_NAME

Returns the NAME that is provided by user at the time of login.

Returns: text_t

NAME

Returns the NAME of the user. This is the name used by user

for login unless it is overriden by name from external

authentication server.

Returns: text_t

PASSWD

Returns the PASSWORD of the user.

Returns: text_t

SESSIONID

Returns the Session ID of the user.

Returns: text_t

ANALYTICS_T

Designates an expression that refers to the analytics. This is the starting point for access

into parameters such as stream.

This topic lists the expressions that are provided by this class.

CONNECTION_QUALITY

Accesses Connection Quality Analytics metrics evaluated for client.

Returns: cqa_t

STREAM(text_tidentifier_name)

Operates on a stream identifier specified in the stream method.

Parameters(expressions not allowed):

identifier_name -

```
Name of the stream identifier added using add
```

stream identifier.

Returns: stream_t

AUTHENTICATION_ERROR_E

This object represents one of the error codes that is encountered

during authentication.

This topic lists the expressions that are provided by this class.

Enumerations

ACCOUNT_DISABLED

This error represents account is disabled for current user.

ACCOUNT_LOCKED

This error represents user's account is locked out.

BAD_PASSWORD

This error represents login attempt for invalid password.

BAD_USER

This error represents login attempt for invalid username.

CHANGE_PASSWORD

This error represents error while trying to change password.

INVALID_CREDS

This error represents invalid credentials presented

for authentication.

INVALID_USER

This error represents username used for login does not exist or

incorrect.

MISMATCH_PASSWORD

This error represents password mismatch for current login attempt.

NOT_PERMITTED

This error represents user is not permitted to login.

NO_DIALIN_PERM

This error represents user does not have dial-in

permissions.

PASSWORD_EXPIRED

This error represents password is expired for the current

user.

RESTRICTED_HOURS

This error represents login attempt in restricted time.

SOCKET_ERROR

This error represents connection error at external authentication server.

SYS_ERROR

This error represents processing error at external authentication server.

SYS_TIMEOUT

This error represents timeout during authentication process

at external authentication server.

Expressions

EQ(authentication_error_ee)

The result is Boolean TRUE if the input argument is equal to

the error presented by this object.

Parameters(expressions not allowed):

e -

```
Authentication Error enumeration
```

Returns: bool_at

NE(authentication_error_ee)

The result is Boolean TRUE if the input argument is not equal to

the error presented by this object.

Parameters(expressions not allowed):

e -

```
Authentication Error enumeration
```

Returns: bool_at

AUTH_KEYID_T

Represents the Authority Key Identifier Extension of the X509V3 certificate.

This topic lists the expressions that are provided by this class.

CERT_SERIALNUMBER

Returns the SerialNumber field of the Authority Key Identifier as a blob.

Returns: text_t

EXISTS

Returns boolean TRUE if the Authority Key Identifier extension exists in the certificate,

otherwise returns boolean FALSE.

Returns: bool_at

ISSUER_NAME

Returns the Issuer Distinguished Name as a '/' seperated Name-Value pair list,

where '=' seperates the Name and the Value.

Example:

/C=IN/ST=KA/L=BLR/O=Netscaler Pvt Ltd/OU=www.netscaler.com/CN=www.netscaler.com/emailAddress=ns@ns.com

Returns: nvlist_t

KEYID

Returns the keyIdentifier field of the Authority Key Identifier as a blob.

Returns: text_t

AVP_T

Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select


"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

AVPLIST_AFTER

Returns the region after this AVP in diameter message (or in other words

returns location at end of this AVP and length covering end

of this AVP to end of diameter message.

If this AVP is inside a Grouped AVP, this method returns region after

this AVP in parent Grouped AVP.

Returns: text_t

AVPLIST_BEFORE

Returns the region till this AVP in diameter message (or in other words

returns location at start of AVPs and length equal to offset of

this AVP).

If this AVP is inside a Grouped AVP, this method returns region till

this AVP in parent AVP.

Returns: text_t

AVP(uint avpcode)

Returns the AVP nested inside the GROUPED AVP (the Data field is a

sequence of AVPs).

For example, to retrieve AVP(263) that is nested in AVP(99999) in a

request, the expression is: DIAMETER.REQ.AVP(999999).AVP(263).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

AVP(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace and that is

nested inside a GROUPED AVP in the message.

For example, to get the AVP with code 1005 and VendorId 10415 from a Request,

that is nested inside 1001 (vendor-id 10415). Use this expression:

DIAMETER.REQ.AVP(1001, 10415).AVP(1004, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

COUNT

Returns the number of occurences of an AVP in diameter message.

The system does not keep track of more than 16 AVP occurences of an AVP. The result

will be undefined if there are more than 16 instances of that AVP.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns true if the AVP exists in the message.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

INSTANCE(uint instanceno)

Returns instances of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

INSTANCE(0) retrieves the first instance, INSTANCE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="avp_t.html">avp_t</a></p></div></div></div><div><div><p><b>* </
b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of boolean type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of double data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 subnet data type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IP_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv4 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_MAC_ADDRESS_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of mac address data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_MANDATORY</b></p><div><p>Returns true if the AVP
has the Mandatory bit set.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of number type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>IS_NUM_AT</b>(<a
href="num_format_e.html">num_format_e</a>format)</p><div><p>Results in
Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>IS_VENDOR_SPECIFIC</b></
p><div><p>Returns true if the AVP has Vendor-Specific bit (V) set.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

LENGTH

Returns the AVP length including AVP header.

If the AVP is not present, it returns length 0. If multiple occurences

of the AVP exist, it returns the length of first occurence.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -
```

```
                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a href="avp_t.html">avp_t</a></
p></div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http%3a//" will be decoded to "http://"
2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the
   ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to
   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW
   denotes the ascii value of character Y in hexadecimal and XX
   denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//"
   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to
   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="avp_t.html">avp_t</a></p></
div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as
DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6
```

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

```
<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -
```

```
                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -
```

quote-character. Returns: nvlist_t * TYPECAST_RADIUS_AVP_T Convert text_t to radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_t : Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert avp_t to
text_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (text_t : Designates any piece
of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from

beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
VALUE

Returns the value of first instance of the AVP as OctetString..

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Alternatively, you can use the expression: DIAMETER.REQ.AVP(282).

Returns: text_t
VALUE(uint instanceno)

Returns value of an instance of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

VALUE(0) retrieves the first instance, VALUE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com

- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>VENDOR_ID</b></p><div><p>Returns the Vendor ID if the
AVP has the Vendor-Specific bit set.</p><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;

- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by
```

```
      /Book/creator"

      in the first 1000 bytes of the body which is

        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost'
```

```
    }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:

  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)
```

```
      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.

    Since this is a node-set the corresponding string

    value will be returned which is "MiltonParadise
```

```
      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is
```

```
      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
```

```
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

AVP_UNUM_T

In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).

For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from avp_t, unsigned_long_at.

*** ABS**

Return absolute double value of the double value.

Returns: double_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

unsigned long.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** AVPLIST_AFTER**

Returns the region after this AVP in diameter message (or in other words

returns location at end of this AVP and length covering end

of this AVP to end of diameter message.

If this AVP is inside a Grouped AVP, this method returns region after

this AVP in parent Grouped AVP.

Returns: text_t

*** AVPLIST_BEFORE**

Returns the region till this AVP in diameter message (or in other words

returns location at start of AVPs and length equal to offset of

this AVP).

If this AVP is inside a Grouped AVP, this method returns region till

this AVP in parent AVP.

Returns: text_t

*** AVP**(uint avpcode)

Returns the AVP nested inside the GROUPED AVP (the Data field is a

sequence of AVPs).

For example, to retrieve AVP(263) that is nested in AVP(99999) in a

request, the expression is: DIAMETER.REQ.AVP(999999).AVP(263).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace and that is

nested inside a GROUPED AVP in the message.

For example, to get the AVP with code 1005 and VendorId 10415 from a Request,

that is nested inside 1001 (vendor-id 10415). Use this expression:

DIAMETER.REQ.AVP(1001, 10415).AVP(1004, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Unsigned Long Value after bitwise negating the current

unsigned long value.

Returns: unsigned_long_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Returns the number of occurences of an AVP in diameter message.

The system does not keep track of more than 16 AVP occurences of an AVP. The result

will be undefined if there are more than 16 instances of that AVP.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EXISTS**

Returns true if the AVP exists in the message.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint instanceno)

Returns instances of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

INSTANCE(0) retrieves the first instance, INSTANCE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="avp_t.html">avp_t</a></p></div></div></div><div><div><p><b>* </
b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of boolean type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of double data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 subnet data type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IP_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv4 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_MAC_ADDRESS_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of mac address data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_MANDATORY</b></p><div><p>Returns true if the AVP has the Mandatory
bit set.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results
in Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_NUM_AT</b>(<a
href="num_format_e.html">num_format_e</a>format)</p><div><p>Results in
Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_VENDOR_SPECIFIC</b></
p><div><p>Returns true if the AVP has Vendor-Specific bit (V) set.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Returns the AVP length including AVP header.

If the AVP is not present, it returns length 0. If multiple occurences

of the AVP exist, it returns the length of first occurence.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LSHIFT**(uint i)

Return the Unsigned Long Value after bitwise left shifting the

current number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>LT</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Results in Boolean TRUE value if the Text object is less than
the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
```

```
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the maximum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Compares the string argument with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the minimum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value that is the renaminder after
dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value that is the renaminder
after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
```

```
p><div><p>Return the Unsigned Long Value after multiplying the argument to
the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after multiplying the
argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Double Value
after negating the current double
```

value.

Returns: double_at

*** NE**(text_ts)

Results in Boolean TRUE value if the Text object is not equal to the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is not equal to
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```


the argument.

Parameters:

i -


```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```


the argument.

Parameters:

i -


```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```


Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Unsigned Long Value after bitwise right shifting the current
```

number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="avp_unum_t.html">avp_unum_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http%3a//" will be decoded to "http://"
2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

For example, the encoded text "http\72//" and "http\072//" will

both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the

ascii value of character Y in hexadecimal.

For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="avp_unum_t.html">avp_unum_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value after subtracting the argument
from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after subtracting the
argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>TYPECAST_AVP_T</b></
p><div><p>Convert avp_unum_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Unsigned Long Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```


treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Typecast the Unsigned Long Value to a Number value.

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Typecast
the Unsigned Long Value to its string representation.</p><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will
raise an undef if the value is null, empty, or the value has an invalid
format for the destination type. (time_at : Treats a number value as
number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

TYPECAST_UNSIGNED_LONG_AT

Convert avp_unum_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
```

```
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9
```

```
Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** VALUE**

Returns the value of first instance of the AVP as OctetString..

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Alternatively, you can use the expression: DIAMETER.REQ.AVP(282).

Returns: text_t
*** VALUE**(uint instanceno)

Returns value of an instance of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

VALUE(0) retrieves the first instance, VALUE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>VENDOR_ID</b></p><div><p>Returns the Vendor ID if the AVP has the
Vendor-Specific bit set.</p><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)


returns the &lt;title&gt; element in the document's


&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the
```

```
  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }
```

```
      The original JSON document is not disturbed but the



      part of the JSON body selected by the



      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'


          }


       },


       "title":'Paradise Lost'


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


    This will return true as there is a node named


    'creator' with parent 'Book' in the first 1000 bytes of the


    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                      XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

BINARY_ENCODING_E

Defines the binary encodings available for the ENCRYPT(), DECRYPT(), and HMAC() functions.

This topic lists the expressions that are provided by this class.

Enumerations

BASE64

Original PEM base64-encoding: 6 bits (0..63) enocded as one ASCII character: 0..23 = 'A'..'Z', 24..51 = 'a'..'z', 52..61 = '0'..'9', 62 = '+', 63 = '/', '=' = pad byte. Four bytes encoded as three characters. CR/LF every 64 characters.

BASE64URL

URL and Filename safe base64-encoding: same as BASE64 except 62 = '-', 63 = '_' and no CR/LFs. Pad byte? Can be used in URLs without requiring escaping.

HEX

Hexadecmimal: For input, accepts HEX_UPPER, HEX_LOWER, and HEX_COLONS format. For output, produces HEX_LOWER format.

HEX_COLONS

Hexadecimal with 0..9 = '0'..'9' and 10..15 = 'A'..'F'; ':' between each hex byte. Matches BLOB_TO_HEX() output format.

HEX_LOWER

Hexadecimal with 0..9 = '0'..'9' and 10..15 = 'a'..'f'.

HEX_UPPER

Hexadecimal with 0..9 = '0'..'9' and 10..15 = 'A'..'F'.

NONE

No encoding: input or output text is raw binary.

BOOL_AT

Boolean Operations

This topic lists the expressions that are provided by this class.

AND(bool_atb)

Logical AND the boolean argument with the current boolean result.

Parameters:

b -

```
Boolean
```

Returns: bool_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

boolean.

Parameters:

str -

```
the string to append.
```

Returns: text_t

NOT

Logical NOT the current boolean result.

Returns: bool_at

OR(bool_atb)

Logical OR the boolean argument with the current boolean result.

Parameters:

b -

```
Boolean
```

Returns: bool_at

TYPECAST_TEXT_T

Typecast the Boolean Value to its string representation.

Returns: text_t

CHARSET_E

Character Set Enum

This topic lists the expressions that are provided by this class.

Enumerations

ASCII

UTF_8

CIPHER_T

SSL CIPHER IN CLIENT HELLO

This topic lists the expressions that are provided by this class.

HAS_HEXCODE(uint code)

Returns true if the cipher stream has the hex code passed in the argument.

Parameters(expressions not allowed):

code -

```
Cipher Hex Code
```

Returns: bool_at

CLIENT_HELLO_T

Client Hello SSL

This topic lists the expressions that are provided by this class.

CIPHERS

Returns the supported ciphers proposed by the client in the client hello message.

Returns: cipher_t

CLIENT_VERSION

Returns the version from the message-header of the client hello message.

Returns: num_at

IS_RENEGOTIATE

Returns true if renegotiation is initiated in ongoing SSL session, else returns false.

Returns: bool_at

IS_REUSE

Returns true if reuse is proposed by client, else returns false.

Returns: bool_at

IS_SCSV

Returns true if in client hello SCSV (Signalling Cipher Suite Value) is advertised, else returns false.

SCSV prevents protocol downgrade attack.

Returns: bool_at

IS_SESSION_TICKET

Returns true if in client hello session ticket is advertised, else returns false.

Returns: bool_at

LENGTH

Returns the length in the message-header of the client hello message.

Returns: num_at

SNI

Returns SNI from CLIENT HELLO

In the case of non-SSL transaction or error in client hello, returns

an empty string.

Returns: ssl_text_t

CLIENT_T

Designates an expression that refers to the client. This is the starting point for access into paramters such as Ether and SSL.

This topic lists the expressions that are provided by this class.

ETHER

Operates on the ethernet protocol data associated with the current packet.

Returns: ether_req_protocol_t

INTERFACE

Designates an expression that refers to the ID of the network interface through which the current packet entered the Application Switch.

Returns: interface_req_t

IP

Operates on the IP protocol data associated with the current packet.

Returns: ip_req_protocol_t

IPV6

Operate on the IPv6 protocol data.

Returns: ipv6_req_protocol_t

SSL

Operates on the SSL protocol data for the current packet.

Returns: ssl_req_protocol_t

SSLVPN

Operates on the SSLVPN data for the current packet.

Returns: sslvpn_req_protocol_t

TCP

Operate on the TCP protocol data.

Returns: tcp_req_protocol_t

TRAFFIC_DOMAIN

Operates on the Traffic Domain through which the current packet entered the Application Switch.

Returns: traffic_domain_req_t

UDP

Operates on the UDP protocol data associated with the current packet.

Returns: udp_req_protocol_t

VLAN

Operates on the VLAN through which the current packet entered the Application Switch.

Returns: vlan_req_t

VXLAN

Operates on the VXLAN through which the current packet entered the Application Switch.

Returns: vxlan_req_t

CQA_T

Retrieves Conection Quality Analytics parameters calculated

for current client.

This topic lists the expressions that are provided by this class.

CONGESTION_LEVEL

Gets detected congestion level.

This returns an integer, in the range 0 to 100, that provides

a measure of network congestion level. The greater the value, the

the higher the detected congestion level is.

Returns: num_at

NETWORK_TYPE

Gets the detected network type.

This should be a string of the detected nework type label

(such as "2G", "3G", "4G", etc.) as defined in cqaparam

configuration (parameters -net1label, -net2label and -net3label).

Returns: text_t

SIGNAL_QUALITY

Gets detected signal quality.

This returns an integer, in the range 0 to 100, that provides

a measure of client signal quality. The greater the value, the

the better the signal quality is.

Returns: num_at

COOKIE_T

The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, nvlist_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Return a Number Value giving the number of name-value

components in the list. If the last Name-Value List

component is of zero length or is all spaces then it is

ignored.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

DOMAIN

Obtain the value of Domain field of the cookie.

If Multiple Domain fields are present the value associated with the

first instance is returned.

For eg:

Set-Cookie : Customer = "ABC"; DOMAIN=".abc.com"; DOMAIN=.xyz.com

then

HTTP.RES.SET_COOKIE.COOKIE("CUSTOMER").DOMAIN gives .abc.com .

Note that the quotes are stripped and value is returned.

String of zero length is returned in the following cases:

```
1) If the Domain field is not present.



2) If the Value of Domain field is not present.
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXPIRES

Obtain the Expires field of the cookie which is date string.

The value of the Expires attribute can be operated upon as a

Time object. The Time object supports a number of date

formats.

If multiple Expires fields are present then the first one is

returned.

If the Expires attribute is absent then Text of length

zero will be returned.

Returns: time_t

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

Based on RFC2396, following are considered URL safe characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

```
Colon: ":"


At symbol: "@"


Question mark: "?"


Equals: "="


Dollar: "$"


Percent: "%"


Ampersand: ";"


Plus: "+"


Comma: ","


Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at

*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
```

```
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAMES</b></p><div><p>Extracts a
list of only the names from the name-value pair list.
```

For example the list 'a=b,c="d=e,f",g=h' will result in the list

'a,c,g'. Empty elements will be ignored.

Returns: list_t
*** NAME**(uint i)

Operate as Text on the name component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(uint i)</
p><div><p>Returns the element in the list identified by the parameter.
```

The parameter is a 0-based index in to the list. The result

is a string. If the specified index is greater than the

number of components then a zero length string is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the name-
value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the name-value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

PATH

Obtain the value of Path field of the cookie, which is a '/'

separated list (Multiple / are treated as single /).

If Multiple path fields are present the value associated with the

first instance is returned.

For eg:

Set-Cookie : Customer = "ABC"; PATH="/a//b/c"; PATH= "/x/y/z"

then

HTTP.RES.SET_COOKIE.COOKIE("CUSTOMER").PATH gives /a//b/c.

HTTP.RES.SET_COOKIE.COOKIE("CUSTOMER").PATH.GET(2) gives b.

Note that the quotes are stripped and value is returned.

String of zero length is returned in the following cases:

```
 1) If the Path field is not present.



 2) If the value of Path field is not present.</p><p><b>Returns: </b><a
href="list_t.html">list_t</a></p></div></div></div><div><div><p><b>* </
b><b>PKEY_ENCRYPT_PEM_NO_PADDING</b>(<a href="text_t.html">text_t</
a>pem_pkey)</p><div><p>Encrypts the input text using the RSA public key
provided by the PEM string
```

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

PORT

Obtain the value of PORT field of the cookie.

Operate as a ',' separated List on the PORT component of the COOKIE.

For eg:

Set-Cookie : Customer = "ABC"; PATH="/a/b/c"; PORT= "80, 2580"

then

HTTP.RES.SET_COOKIE.COOKIE("ABC").PORT gives 80, 2580.

Note that the quotes are stripped and value is returned.

String of zero length is returned in the following cases:

```
 1) If the Port field is not present.



 2) If the value of Port field is not present.</p><p><b>Returns: </b><a
href="list_t.html">list_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(char c, uint i)</p><div><p>Select a prefix of the Text.
The selected prefix is the
```

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>REPLACE_ALL_VALUES</b>(<a
href="text_t.html">text_t</a>value)</p><div><p>Replaces all the values in
each name-value pair with the specified
```

string argument.

For example, given argument string "?" the list 'a=b,c=d,e=f' will

result in the name-value list 'a=?,c=?,e=?'. Quotes will be stripped

from the values being replaced. Empty values will be replaced with

the value provided. Empty elements will be preserved.

Parameters(expressions not allowed):

value -

```
value to replace existing values</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```


use text.

Parameters(expressions not allowed):

charset -


```
Character Set</p></div><p><b>Returns: </b><a
href="cookie_t.html">cookie_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_TEXT_MODE</b>(<a href="textmode_e.html">textmode_e</a>mode)</
p><div><p>The mode setting of a Text object controls the behavior of
```


many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="cookie_t.html">cookie_t</
a></p></div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint
i)</p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.


<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.


Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert cookie_t
to text_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (text_t : Designates any
piece of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUES**

Extracts a list of only the values from the name-value pair list.

For example the list 'a=b,c=d,e=f' will result in the list

'b,d,f'. Note that lists do not honor the quote characters, so

any nvlist_at entry that has quotes and a quoted entry includes a

separator will be split at that embedded separator. Quote

characters will not be stripped from the values. Empty elements

and empty values will be ignored.

Returns: list_t
*** VALUE**(uint i)

Operate as Text on the value component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the value
component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the value component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>VERSION</b></p><div><p>Obtain the Value of Version
field of the cookie which is a
```

decimal integer. If Multiple version fields are present the

value associated with the first instance after the COOKIE

name-value pair is returned.

For eg: Set-Cookie : Customer = "ABC"; VERSION = "1";

VERSION = "0" then

HTTP.RES.SET_COOKIE.COOKIE("CUSTOMER").VERSION gives 1.

Note that the quotes are stripped and value is returned.

Default numerical value zero is returned in the following cases:

1. If the version field is not present. 2) If the value of

version field is not present. 3) If the version value is not

a decimal number.

Returns: num_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }

    The original JSON document is not disturbed but the
```

```
      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{


            "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


    This will return true as there is a node named


    'creator' with parent 'Book' in the first 1000 bytes of the


    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
```

```
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

DIAMETER_FLAG_E

This object represents one of the diameter flags.

This topic lists the expressions that are provided by this class.

Enumerations

ERROR

PROXIABLE

REQUEST

RESERVED_4

RESERVED_5

RESERVED_6

RESERVED_7

RETRANSMIT

DIAMETER_FLAG_T

Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
                        number format</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_SET</b>(<a
href="diameter_flag_e.html">diameter_flag_e</a>flag)</p><div><p>Test a
flag bit (REQUEST, PROXIABLE, ERROR or RETRANSMIT) in diameter message.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>flag -



                        flag bit</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_STRINGMAP_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns true if the string that it operates
on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>RESET_ALL</b></p><div><p>Reset flag bits in diameter
message. Returns diameter_flag_t which
```

allows further set/unset of other bits in this flag

Returns: diameter_flag_t

*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="diameter_flag_t.html">diameter_flag_t</a></p></div></div></
div><div><div><p><b>SET</b>(<a
href="diameter_flag_e.html">diameter_flag_e</a>flag)</p><div><p>Set a flag
bit (REQUEST, PROXIABLE, ERROR or RETRANSMIT) in diameter message.
```

Returns diameter_flag_t which allows further set/unset of other bits

in this flag

Parameters(expressions not allowed):

flag -

```
flag bit</p></div><p><b>Returns: </b><a
href="diameter_flag_t.html">diameter_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

For example, the encoded text "http%3a//" will be decoded to "http://"

2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="diameter_flag_t.html">diameter_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
```

```
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -
```

```
                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.
```

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

```
ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
```

```
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
```

```
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
diameter_flag_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
```

```
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>UNSET</b>(<a
href="diameter_flag_e.html">diameter_flag_e</a>flag)</p><div><p>Unset a
flag bit (REQUEST, PROXIABLE, ERROR or RETRANSMIT) in diameter message.
```

Returns diameter_flag_t which allows further set/unset of other bits

in this flag

Parameters(expressions not allowed):

flag -

```
flag bit</p></div><p><b>Returns: </b><a
href="diameter_flag_t.html">diameter_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
```

```
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9
```

```
Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
   HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



   returns the &lt;title&gt; element in the document's



   &lt;head&gt; This can be used in a rewrite action



   to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)


  returns true if the &lt;meta&gt; element exists as a child


  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)


  returns the numeric value of the height attribute of the
```

```
  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)
```

This expression when operated on the above JSON file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

```
  creator:{

      person:{

        name:'Milton'

      }

  }
```

The original JSON document is not disturbed but the

```
      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{


            "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                          XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
```

```
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

DIAMETER_MESSAGE_T

Define common operations for Diameter.Req or Diameter.Res.

This topic lists the expressions that are provided by this class.

ACCOUNTING_SUB_SESSION_ID

Returns the Accounting-Sub-Session-Id AVP (code 287) from the message.

Returns: avp_t

ACCT_APPLICATION_ID

Returns the Acct-Application-Id AVP (code 259) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

ACCT_MULTI_SESSION_ID

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: avp_t

ACCT_SESSION_ID

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: avp_t

APPLICATION_ID

Returns the Application-ID from the message.

Returns: unum_text_t

AUTH_APPLICATION_ID

Returns the Auth-Application-Id AVP (code 258) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

AVP_END

Returns the location at the end of Diameter message (or in other words

end of all AVPs) in Diameter message.

Returns: text_t

AVP_LIST

Returns the AVP region in Diameter message (or in other words

returns location at start of AVPs and length equal to Diameter

message length excluding header).

Returns: text_t

AVP(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 263 from a Request, use the expression:

DIAMETER.REQ.AVP(263).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

AVP(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace from the message.

For example, to get the AVP with code 1001 and VendorId 10415 from a Request,

use the expression: DIAMETER.REQ.AVP(1001, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

AVP_START

Returns the location at the start of AVPs (or in other words

end of diameter header) in Diameter message.

Returns: text_t

COMMAND_CODE

Returns the Command Code from the message.

Returns: unum_text_t

END_TO_END_ID

Returns the End-to-End Identifier from the message.

Returns: unsigned_long_at

FLAGS

Returns the value of Diameter Header flags in Diameter message.

Returns: diameter_flag_t

HOP_BY_HOP_ID

Returns the Hop-by-Hop Identifier from the message.

Returns: unsigned_long_at

IS_PROXIABLE

Returns true if the P (proxiable) command flag is set for the message.

Returns: bool_at

LENGTH

Returns the Message Length from the message.

Returns: num_at

ORIGIN_HOST

Returns the Origin-Host AVP (code 264) from the message.

Returns: avp_t

ORIGIN_REALM

Returns the Origin-Realm AVP (code 296) from the message.

Returns: avp_t

PRODUCT_NAME

Returns the Product-Name AVP (code 269) from the message.

Returns: avp_t

SESSION_ID

Returns the Session-ID AVP (code 263) from the message.

Returns: avp_t

SUPPORTED_VENDOR_ID

Returns the Supported-Vendor-Id AVP (code 265) from the message.

Returns: avp_t

USER_NAME

Returns the User-Name AVP (code 1) from the message.

Returns: avp_t

VENDOR_ID

Returns the Vendor-Id AVP (code 266) from the message.

Returns: avp_t

VENDOR_SPECIFIC_APPLICATION_ID

Returns the Vendor-Specific-Application-Id AVP (code 260) from the message.

Returns: vendor_spec_appid_avp_t

VERSION

Returns the version of the Diameter protocol. Currently, Diameter supports

only version 1.

Returns: unum_text_t

DIAMETER_PROTOCOL_T

Provides expressions to retrieve Diameter header information and Diameter

Header and AVPs (attribute value pairs) from Diameter requests and responses.

Note: The Diameter implementation is RFC 6733 compliant.

This topic lists the expressions that are provided by this class.

IS_CLIENT

Returns true if this is a Diameter client connection.

Returns: bool_at

IS_SERVER

Returns true if this is a Diameter server connection.

Returns: bool_at

NEW_ANSWER

Creates new Diameter Response Message (Header only) based on Request header.

Returns: text_t

NEW_ANSWER(bool_aterror, ip_address_atcommand_code, ip_address_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, ip_address_atcommand_code, num_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, ip_address_atcommand_code, time_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, ip_address_atcommand_code, unsigned_long_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, num_atcommand_code, ip_address_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, num_atcommand_code, num_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, num_atcommand_code, time_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, num_atcommand_code, unsigned_long_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, time_atcommand_code, ip_address_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, time_atcommand_code, num_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, time_atcommand_code, time_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, time_atcommand_code, unsigned_long_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, unsigned_long_atcommand_code, ip_address_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, unsigned_long_atcommand_code, num_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, unsigned_long_atcommand_code, time_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_ANSWER(bool_aterror, unsigned_long_atcommand_code, unsigned_long_atapplication_id)

Creates new Diameter Message (Header only) based on given parameters.

Parameters:

error -

```
Error flag
```

command_code -

```
Command associated with the Diameter message
```

application_id -

```
Application id to be used in Diameter Header
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, double_atavpvalue)

Creates new AVP based on given avp code and given Float32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given Float32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Float32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, time_atavpvalue)

Creates new AVP based on given avp code and given Float32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, unsigned_long_atavpvalue)

Creates new AVP based on given avp code and given Float32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, double_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, ip_address_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, num_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, time_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT32(uint avpcode, unsigned_long_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, double_atavpvalue)

Creates new AVP based on given avp code and given Float64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given Float64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Float64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, time_atavpvalue)

Creates new AVP based on given avp code and given Float64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, unsigned_long_atavpvalue)

Creates new AVP based on given avp code and given Float64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, double_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, ip_address_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, num_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, time_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_FLOAT64(uint avpcode, unsigned_long_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Float64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_INTEGER32(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Integer32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER32(uint avpcode, num_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Integer32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given Integer64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Integer64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, time_atavpvalue)

Creates new AVP based on given avp code and given Integer64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, unsigned_long_atavpvalue)

Creates new AVP based on given avp code and given Integer64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, ip_address_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Integer64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, num_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Integer64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, time_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Integer64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, unsigned_long_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Integer64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP(uint avpcode, text_tavpvalue)

Creates new AVP based on given avp code and given value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value as octet string
```

Returns: text_t

NEW_AVP(uint avpcode, text_tavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given value. Set

the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value as octet string
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_UNSIGNED32(uint avpcode, uint avpvalue)

Creates new AVP based on given avp code and given Unsigned32 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_UNSIGNED32(uint avpcode, uint avpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Unsigned32 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given Unsigned64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Unsigned64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, time_atavpvalue)

Creates new AVP based on given avp code and given Unsigned64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, unsigned_long_atavpvalue)

Creates new AVP based on given avp code and given Unsigned64 value.

Vendor-Specific and Mandatory bit are not set.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, ip_address_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Unsigned64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, num_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Unsigned64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, time_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Unsigned64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_AVP_UNSIGNED64(uint avpcode, unsigned_long_atavpvalue, bool_atis_mandatory, uint vendor_id)

Creates new AVP based on given avp code and given Unsigned64 value.

Set the 'M' and 'V' bits, and vendor_id field for this AVP.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

is_mandatory -

```
'true' to set The 'M' bit in AVP
```

vendor_id -

```
vendor_id field. non-zero vendor_id sets
```

the 'V' bit and fills the vendor_id field in AVP as

given.

Returns: text_t

NEW_ERROR_ANSWER

Creates new Diameter Error Response Message (Header only) based on Request header.

Returns: text_t

NEW_REDIRECT(text_tredirectUri)

Creates new Diameter Redirect Response Message (Header only) based on Request header.

This Redirect message will not have Redirect-Host-Usage and

Redirect-Max-Cache-Time AVP.

Parameters(expressions not allowed):

redirectUri -

```
Redirect-Host AVP value
```

Returns: text_t

NEW_REDIRECT(text_tredirectUri, uint hostUsage, uint maxCacheTime)

Creates new Diameter Redirect Response Message (Header only) based on Request header.

Use hostUsage and maxCacheTime parameters to insert

Redirect-Host-Usage and Redirect-Max-Cache-Time AVP

as defined in RFC 6733 section-6.13 (http://tools.ietf.org/html/rfc6733#section-6.13)

Parameters(expressions not allowed):

redirectUri -

```
Redirect-Host AVP value
```

hostUsage -

```
Redirect-Host-Usage AVP value
```

maxCacheTime -

```
Redirect-Max-Cache-Time AVP value.
```

Returns: text_t

REQ

Retrieves header information and AVPs from Diameter requests.

Returns: diameter_req_t

RES

Retrieves header information and AVPs from Diameter responses.

Returns: diameter_res_t

DIAMETER_REQ_T

Retrieves header information and AVPs from Diameter requests.

AVPs can be retrieved by:

- Directly specifying the AVP codes in the form: DIAMETER.REQ.AVP(<avpcode>). For example, to retrieve the Auth-Application-Id AVP (code 258), you can use the expression: DIAMETER.REQ.AVP(258).
- Using aliases that are provided for important AVPs. For example, the Auth-Application-Id AVP has the alias AUTH_APPLICATION_ID. So, the expression to retrieve the Auth-Application-Id AVP by using the alias is DIAMETER.REQ.AUTH_APPLICATION_ID.

General usage:DIAMETER.REQ.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from diameter_message_t.

*** ACCOUNTING_SUB_SESSION_ID**

Returns the Accounting-Sub-Session-Id AVP (code 287) from the message.

Returns: avp_t

*** ACCT_APPLICATION_ID**

Returns the Acct-Application-Id AVP (code 259) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** ACCT_MULTI_SESSION_ID**

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: avp_t

*** ACCT_SESSION_ID**

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: avp_t

*** APPLICATION_ID**

Returns the Application-ID from the message.

Returns: unum_text_t

*** AUTH_APPLICATION_ID**

Returns the Auth-Application-Id AVP (code 258) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** AVP_END**

Returns the location at the end of Diameter message (or in other words

end of all AVPs) in Diameter message.

Returns: text_t

*** AVP_LIST**

Returns the AVP region in Diameter message (or in other words

returns location at start of AVPs and length equal to Diameter

message length excluding header).

Returns: text_t

*** AVP**(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 263 from a Request, use the expression:

DIAMETER.REQ.AVP(263).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace from the message.

For example, to get the AVP with code 1001 and VendorId 10415 from a Request,

use the expression: DIAMETER.REQ.AVP(1001, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

*** AVP_START**

Returns the location at the start of AVPs (or in other words

end of diameter header) in Diameter message.

Returns: text_t

*** COMMAND_CODE**

Returns the Command Code from the message.

Returns: unum_text_t

DESTINATION_HOST

Returns the Destination-Host AVP (code 293) from the request.

Returns: avp_t

DESTINATION_REALM

Returns the Destination-Realm AVP (code 283) from the request.

Returns: avp_t

*** END_TO_END_ID**

Returns the End-to-End Identifier from the message.

Returns: unsigned_long_at

*** FLAGS**

Returns the value of Diameter Header flags in Diameter message.

Returns: diameter_flag_t

*** HOP_BY_HOP_ID**

Returns the Hop-by-Hop Identifier from the message.

Returns: unsigned_long_at

*** IS_PROXIABLE**

Returns true if the P (proxiable) command flag is set for the message.

Returns: bool_at

IS_RETRANSMIT

Returns true if the T (potentially retransmitted message) command flag is

set for the request.

Returns: bool_at

*** LENGTH**

Returns the Message Length from the message.

Returns: num_at

*** ORIGIN_HOST**

Returns the Origin-Host AVP (code 264) from the message.

Returns: avp_t

*** ORIGIN_REALM**

Returns the Origin-Realm AVP (code 296) from the message.

Returns: avp_t

*** PRODUCT_NAME**

Returns the Product-Name AVP (code 269) from the message.

Returns: avp_t

*** SESSION_ID**

Returns the Session-ID AVP (code 263) from the message.

Returns: avp_t

*** SUPPORTED_VENDOR_ID**

Returns the Supported-Vendor-Id AVP (code 265) from the message.

Returns: avp_t

TYPECAST_DIAMETER_MESSAGE_T

Convert diameter_req_t to diameter_message_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_message_t : Define common operations for Diameter.Req or Diameter.Res.)

Returns: diameter_message_t

*** USER_NAME**

Returns the User-Name AVP (code 1) from the message.

Returns: avp_t

*** VENDOR_ID**

Returns the Vendor-Id AVP (code 266) from the message.

Returns: avp_t

*** VENDOR_SPECIFIC_APPLICATION_ID**

Returns the Vendor-Specific-Application-Id AVP (code 260) from the message.

Returns: vendor_spec_appid_avp_t

*** VERSION**

Returns the version of the Diameter protocol. Currently, Diameter supports

only version 1.

Returns: unum_text_t

DIAMETER_RES_T

Retrieves header information and AVPs from Diameter responses.

AVPs can be retrieved by:

- Directly specifying the AVP codes in the form: DIAMETER.RES.AVP(<avpcode>). For example, to retrieve the Auth-Application-Id AVP (code 258), you can use the expression: DIAMETER.RES.AVP(258).
- Using aliases that are provided for important AVPs. For example, the Auth-Application-Id AVP has the alias AUTH_APPLICATION_ID. So, the expression to retrieve the Auth-Application-Id AVP by using the alias is DIAMETER.RES.AUTH_APPLICATION_ID.

General usage:DIAMETER.RES.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from diameter_message_t.

*** ACCOUNTING_SUB_SESSION_ID**

Returns the Accounting-Sub-Session-Id AVP (code 287) from the message.

Returns: avp_t

*** ACCT_APPLICATION_ID**

Returns the Acct-Application-Id AVP (code 259) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** ACCT_MULTI_SESSION_ID**

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: avp_t

*** ACCT_SESSION_ID**

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: avp_t

*** APPLICATION_ID**

Returns the Application-ID from the message.

Returns: unum_text_t

*** AUTH_APPLICATION_ID**

Returns the Auth-Application-Id AVP (code 258) from the message.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** AVP_END**

Returns the location at the end of Diameter message (or in other words

end of all AVPs) in Diameter message.

Returns: text_t

*** AVP_LIST**

Returns the AVP region in Diameter message (or in other words

returns location at start of AVPs and length equal to Diameter

message length excluding header).

Returns: text_t

*** AVP**(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 263 from a Request, use the expression:

DIAMETER.REQ.AVP(263).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace from the message.

For example, to get the AVP with code 1001 and VendorId 10415 from a Request,

use the expression: DIAMETER.REQ.AVP(1001, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

*** AVP_START**

Returns the location at the start of AVPs (or in other words

end of diameter header) in Diameter message.

Returns: text_t

*** COMMAND_CODE**

Returns the Command Code from the message.

Returns: unum_text_t

*** END_TO_END_ID**

Returns the End-to-End Identifier from the message.

Returns: unsigned_long_at

*** FLAGS**

Returns the value of Diameter Header flags in Diameter message.

Returns: diameter_flag_t

*** HOP_BY_HOP_ID**

Returns the Hop-by-Hop Identifier from the message.

Returns: unsigned_long_at

IS_ERROR

Returns true if the E (error) command flag is set for the response.

Returns: bool_at

*** IS_PROXIABLE**

Returns true if the P (proxiable) command flag is set for the message.

Returns: bool_at

*** LENGTH**

Returns the Message Length from the message.

Returns: num_at

*** ORIGIN_HOST**

Returns the Origin-Host AVP (code 264) from the message.

Returns: avp_t

*** ORIGIN_REALM**

Returns the Origin-Realm AVP (code 296) from the message.

Returns: avp_t

*** PRODUCT_NAME**

Returns the Product-Name AVP (code 269) from the message.

Returns: avp_t

REDIRECT_HOST

Returns the Redirect-Host AVP (code 292) from the response.

Returns: avp_t

RESULT_CODE

Returns the Result-Code AVP (code 268) from the response.

Returns: avp_t

*** SESSION_ID**

Returns the Session-ID AVP (code 263) from the message.

Returns: avp_t

*** SUPPORTED_VENDOR_ID**

Returns the Supported-Vendor-Id AVP (code 265) from the message.

Returns: avp_t

TYPECAST_DIAMETER_MESSAGE_T

Convert diameter_res_t to diameter_message_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_message_t : Define common operations for Diameter.Req or Diameter.Res.)

Returns: diameter_message_t

*** USER_NAME**

Returns the User-Name AVP (code 1) from the message.

Returns: avp_t

*** VENDOR_ID**

Returns the Vendor-Id AVP (code 266) from the message.

Returns: avp_t

*** VENDOR_SPECIFIC_APPLICATION_ID**

Returns the Vendor-Specific-Application-Id AVP (code 260) from the message.

Returns: vendor_spec_appid_avp_t

*** VERSION**

Returns the version of the Diameter protocol. Currently, Diameter supports

only version 1.

Returns: unum_text_t

DIGEST_METHOD_E

Defines the message digest methods allowed in DIGEST().

This topic lists the expressions that are provided by this class.

Enumerations

MD2

MD4

MD5

SHA1

SHA224

SHA256

SHA384

SHA512

DNS_DOMAINNAME_T

This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

EQ(text_thostname)

Operates on the text contents of domainname. Results in Boolean TRUE

value if the domainname matches the input argument.

The comparison will be case-insensitive EQ(www.forbar.com) returns

TRUE.

Parameters:

hostname -

```
Hostname
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="dns_domainname_t.html">dns_domainname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="dns_domainname_t.html">dns_domainname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

Format Returns: num_t * TYPECAST_NUM_T(num_format_eformat, num_atdefault_value) Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

Format default_value -

Number Returns: num_t * TYPECAST_NVLIST_T(char sep1, char sep2) Convert text_t to nvlist_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (nvlist_t : Two character delimiters have to be provided when

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
dns_domainname_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

> identifies the certkey object containing the RSA

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

DNS_ECS_T

This object provides operations for parsing the DNS ECS option.

This topic lists the expressions that are provided by this class.

IP

Returns the representative IPv4 address of the 'address' field

in EDNS client subnet.

Returns: ip_address_at

IPV6

Returns the representative IPv6 address of the 'address' field

in EDNS client subnet.

Returns: ipv6_address_at

IS_IPV4

Returns true if EDNS client subnet option contains an IPv4

address

Returns: bool_at

IS_IPV6

Returns true if EDNS client subnet option contains an IPv6

address

Returns: bool_at

DNS_FLAG_E

This object represents one of the DNS header flags.

This topic lists the expressions that are provided by this class.

Enumerations

AA

AD

CD

QR

RA

RD

RESERVED

TC

DNS_FLAG_T

Provides operations to test/set/unset a flag in dns header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

GET_STRING_REPRESENTATION

Return all the flag bits set in the DNS header in string

format

Returns: text_t

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
                        number format</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_SET</b>(<a href="dns_flag_e.html">dns_flag_e</
a>flag)</p><div><p>Test a flag bit  in dns header.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>flag -



                        bit</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_STRINGMAP_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns true if the string that it operates
on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
```

```
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```


use text.

Parameters(expressions not allowed):

charset -


```
                        Character Set</p></div><p><b>Returns: </b><a
href="dns_flag_t.html">dns_flag_t</a></p></div></div></
div><div><div><p><b>SET</b>(<a href="dns_flag_e.html">dns_flag_e</
a>flag)</p><div><p>Set a flag bit in the DNS header. Returns dns_flag_t</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>flag -




                        bit</p></div><p><b>Returns: </b><a
href="dns_flag_t.html">dns_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```


many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

"http://"

2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="dns_flag_t.html">dns_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.
```

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

```
ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
```

```
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
```

```
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
dns_flag_t to text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
```

```
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>UNSET</
b>(<a href="dns_flag_e.html">dns_flag_e</a>flag)</p><div><p>Unset a flag
bit in the DNS header. Returns dns_flag_t</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>flag -



                        bit</p></div><p><b>Returns: </b><a
href="dns_flag_t.html">dns_flag_t</a></p></div></div></
div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

```
    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

> identifies the certkey object containing the RSA

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

> identifies the certkey object containing an X.509

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)
```

```
This expression when operated on the above JSON file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

  creator:{

     person:{

       name:'Milton'

     }

  }

The original JSON document is not disturbed but the

part of the JSON body selected by the
```

```
      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'



           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.
```

```
      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)


      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"
```

```
      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)


      This will return true as there is a node called creator
```

```
      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

DNS_HEADER_T

This object provides operations for parsing the DNS header.

This topic lists the expressions that are provided by this class.

ANCOUNT

Returns the answer count in the DNS header.

Returns: num_at

ARCOUNT

Returns the additional record count in the DNS header.

Returns: num_at

FLAGS

Returns the value of DNS header flags

Returns: dns_flag_t

NSCOUNT

Returns the nameserver count in the DNS header.

Returns: num_at

OPCODE

Return the opcode in the DNS header.

Returns: dns_opcode_t

QDCOUNT

Returns the question count in the DNS header.

Returns: num_at

RCODE

Return the response code of the DNS question

Returns: dns_rcode_t

DNS_OPCODE_E

This object provides operations for parsing the DNS header opcode.

This topic lists the expressions that are provided by this class.

Enumerations

IQUERY

NOTIFY

QUERY

STATUS

UPDATE

Expressions

EQ(dns_opcode_em)

The result is Boolean TRUE if the input argument is equal to

the DNS opcode type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns opcode type
```

Returns: bool_at

NE(dns_opcode_em)

The result is Boolean FALSE if the input argument is equal to

the DNS opcode type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns opcode type
```

Returns: bool_at

DNS_OPCODE_T

This object provides Num operations and Dns OpcodeType Enum

operations.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from dns_opcode_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(dns_opcode_em)

The result is Boolean TRUE if the input argument is equal to

the DNS opcode type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns opcode type
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(dns_opcode_em)

The result is Boolean FALSE if the input argument is equal to

the DNS opcode type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns opcode type
```

Returns: bool_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

SET(dns_opcode_eopcode)

Sets the specified opcode in the DNS header

Parameters(expressions not allowed):

opcode -

```
enum value
```

Returns: dns_opcode_t

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert dns_opcode_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

DNS_OPT_T

This object provides operations for parsing the DNS OPT record.

This topic lists the expressions that are provided by this class.

ECS

Returns EDNS client subnet option (if exists).

Returns: dns_ecs_t

ECS_EXISTS

Returns true if EDNS client subnet option is present in the

OPT record.

Returns: bool_at

DNS_PROTOCOL_T

Models the DNS message types.

This topic lists the expressions that are provided by this class.

LENGTH

Returns the DNS packet length.

In case of DNS-TCP, it includes 2 byte length field as well.

Returns: num_at

NEW_RESPONSE

Creates an empty DNS response based on request header.

Returns: text_t

NEW_RESPONSE(bool_atAA, bool_atTC, dns_rcode_ercode)

Creates a new empty DNS response based on the given parameters.

Parameters(expressions not allowed):

AA -

```
Authoritative Answer flag
```

TC -

```
Truncated flag
```

rcode -

```
Retrun Code
```

Returns: text_t

NEW_RRSET_AAAA(text_tiplist, uint ttl)

Creates a new RR set with the given ipv6 address and ttl

parameters.

Parameters(expressions not allowed):

iplist -

```
Comma separated IPv6 address list
```

ttl -

```
Time To Live
```

Returns: text_t

NEW_RRSET_A(text_tiplist, uint ttl)

Creates a new RR set with the given ipv4 address and ttl

parameters.

Parameters(expressions not allowed):

iplist -

```
Comma separated IP address list
```

ttl -

```
Time To Live
```

Returns: text_t

REQ

Operates on DNS Request.

Returns: dns_req_t

RES

Operates on DNS Response.

Returns: dns_res_t

RR

Operates on DNS Resource Record(RR).

Returns: dns_rr_t

DNS_QUESTION_T

This object provides operations for parsing the DNS question.

This topic lists the expressions that are provided by this class.

DOMAIN

Returns domain name.

Returns: dns_domainname_t

TYPE

Returns DNS Record type.

Returns: dns_type_t

DNS_RCODE_E

This object provides operations for parsing the DNS header response code.

This topic lists the expressions that are provided by this class.

Enumerations

FORMERR

NOERROR

NOTIMP

NXDOMAIN

REFUSED

SERVFAIL

Expressions

EQ(dns_rcode_em)

The result is Boolean TRUE if the input argument is equal to

the DNS response code represented by this object.

Parameters(expressions not allowed):

m -

```
DNS response code
```

Returns: bool_at

NE(dns_rcode_em)

The result is Boolean TRUE if the input argument is not equal to

the DNS response code represented by this object.

Parameters(expressions not allowed):

m -

```
DNS response code
```

Returns: bool_at

DNS_RCODE_T

This object provides Num operations and Dns RcodeType Enum

operations.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from dns_rcode_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(dns_rcode_em)

The result is Boolean TRUE if the input argument is equal to

the DNS response code represented by this object.

Parameters(expressions not allowed):

m -

```
DNS response code
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(dns_rcode_em)

The result is Boolean TRUE if the input argument is not equal to

the DNS response code represented by this object.

Parameters(expressions not allowed):

m -

```
DNS response code
```

Returns: bool_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

SET(dns_rcode_ercode)

Sets the specified rcode in the DNS header

Parameters(expressions not allowed):

rcode -

```
enum value
```

Returns: dns_rcode_t

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert dns_rcode_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

DNS_RDATA_T

This object provides operations for interpreting the data present in RDATA section of DNS Resource Record (RR).

This topic lists the expressions that are provided by this class.

IP

Returns the IP address from an A record.

Returns: ip_address_at

IPV6

Returns the IPV6 address from an AAAA record.

Returns: ipv6_address_at

DNS_REQ_T

Operates on DNS Request.

This is the starting point for accessing request side dns packet

This topic lists the expressions that are provided by this class.

HEADER

Returns DNS request header information.

Returns: dns_header_t

OPT

Returns the OPT record (if it exists) in the additional

section.

Returns: dns_opt_t

QUESTION

Returns DNS question which has domain and resource record

information.

Returns: dns_question_t

TRANSPORT

Returns DNS transport layer protocol type.

Returns: dns_transport_e

DNS_RES_T

Operates on DNS Response.

This is the starting point for accessing response side dns packet

This topic lists the expressions that are provided by this class.

HEADER

Returns DNS response header which contains response code

information.

Returns: dns_header_t

OPT

Returns the OPT record (if it exists) in the additional

section.

Returns: dns_opt_t

QUESTION

Returns DNS question which has domain and resource record

information.

Returns: dns_question_t

TRANSPORT

Returns DNS transport layer protocol type.

Returns: dns_transport_e

DNS_RR_T

This object provides operations for parsing the DNS Resource Record (RR).

This topic lists the expressions that are provided by this class.

RDATA

Returns RDATA in the RR.

Returns: dns_rdata_t

TYPE

Returns The DNS Record type.

Returns: dns_type_e

DNS_TRANSPORT_E

This object provides DNS Transport information

This topic lists the expressions that are provided by this class.

Enumerations

TCP

UDP

Expressions

EQ(dns_transport_em)

The result is Boolean TRUE if the input argument is equal to

the DNS transport protocol type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns transport protocol type
```

Returns: bool_at

NE(dns_transport_em)

The result is Boolean FALSE if the input argument is equal to

the DNS transport protocol type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns transport protocol type
```

Returns: bool_at

DNS_TYPE_E

This object provides operations for parsing the DNS resourse record type.

This topic lists the expressions that are provided by this class.

Enumerations

A

AAAA

ANY

CNAME

MX

NAPTR

NS

PTR

SOA

SRV

TXT

Expressions

EQ(dns_type_em)

The result is Boolean TRUE if the input argument is equal to

the DNS record type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns record type
```

Returns: bool_at

NE(dns_type_em)

The result is Boolean FALSE if the input argument is equal to

the DNS record type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns record type
```

Returns: bool_at

DNS_TYPE_T

This object provides Num operations and DnsType Enum

operations on the DNS Record Type.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from dns_type_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(dns_type_em)

The result is Boolean TRUE if the input argument is equal to

the DNS record type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns record type
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(dns_type_em)

The result is Boolean FALSE if the input argument is equal to

the DNS record type represented by this object.

Parameters(expressions not allowed):

m -

```
Dns record type
```

Returns: bool_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert dns_type_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

DOUBLE_AT

Double Value

This topic lists the expressions that are provided by this class.

ABS

Return absolute double value of the double value.

Returns: double_at

ADD(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

ADD(ip_address_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

ADD(num_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

ADD(time_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

ADD(unsigned_long_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

double.

Parameters:

str -

```
the string to append.
```

Returns: text_t

BETWEEN(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

DIV(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

DIV(ip_address_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

DIV(num_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

DIV(time_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

DIV(unsigned_long_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

EQ(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(ip_address_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(num_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(time_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(unsigned_long_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(ip_address_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(num_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(time_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(unsigned_long_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(ip_address_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(num_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(time_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(unsigned_long_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(ip_address_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(num_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(time_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(unsigned_long_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(ip_address_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(num_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(time_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(unsigned_long_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

MAX(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MAX(ip_address_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MAX(num_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MAX(time_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MAX(unsigned_long_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(ip_address_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(num_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(time_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(unsigned_long_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(ip_address_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(num_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(time_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(unsigned_long_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

NEG

Return the Double Value after negating the current double

value.

Returns: double_at

NE(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

NE(ip_address_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

NE(num_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

NE(time_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

NE(unsigned_long_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

SUB(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

SUB(ip_address_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

SUB(num_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

SUB(time_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

SUB(unsigned_long_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

TYPECAST_NUM_AT

Typecast the Double Value to a Number value.

Returns: num_at

TYPECAST_TEXT_T

Typecast the Double Value to its string representation.

Returns: text_t

TYPECAST_UNSIGNED_LONG_AT

Typecast the Double Value to a Unsigned Long value.

Returns: unsigned_long_at

DOUBLE_FORMAT_E

Double Format Enum

This topic lists the expressions that are provided by this class.

Enumerations

DECIMAL

Decimal representation of a positive or a negative double precisions floating point value. White spaces before and after the number are OK

DECIMAL_PREFIX

Decimal representation of a positive or a negative double precisions floating point value followed by arbitrary characters. Examples " -123.45" " 234e5 " "- 123e5abc" "234e5 abc"

DOUBLE_T

Provides operations to treat the underlying text as a Double

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, double_at.

*** ABS**

Return absolute double value of the double value.

Returns: double_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(num_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(time_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(unsigned_long_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select


"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

double.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

> the time value to append.

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(num_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(time_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(unsigned_long_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Return the maximum double value
after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the minimum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
```

```
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Return the minimum double value
after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after multiplying the argument to the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the Double
Value after multiplying the argument to the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Return the Double Value after
multiplying the argument to the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Double Value after
multiplying the argument to the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Double Value after multiplying the argument to the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return
the Double Value after negating the current double
```

value.

Returns: double_at
*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the double value is not equal to
```


the argument.

Parameters:

i -


```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is not equal to
```


the argument.

Parameters:

i -


```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
double value is not equal to
```


the argument.

Parameters:

i -


```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is not equal to
```


the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_TEXT_MODE</b>(<a href="textmode_e.html">textmode_e</a>mode)</
p><div><p>The mode setting of a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="double_t.html">double_t</
a></p></div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint
i)</p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the Double
Value after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Return the Double Value after
subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Double Value after
subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Double Value after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint
i)</p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
TYPECAST_DOUBLE_AT

Convert double_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.
```

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

```
ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Typecast the Double Value to a Number value.

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
```

```
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
```

```
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert double_t
to text_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (text_t : Designates any
piece of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Double Value to a Unsigned Long value.

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)
```

```
  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"
```

```
      in the first 1000 bytes of the body which is


        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is
```

```
      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)


      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

ENCRYPT_METHOD_E

Defines the encryption methods allowed in XML_ENCRYPT().

This topic lists the expressions that are provided by this class.

Enumerations

AES128

AES192

AES256

DES3

RC4

ENDIAN_E

Endianness Enum

This topic lists the expressions that are provided by this class.

Enumerations

BIG_ENDIAN

LITTLE_ENDIAN

ERROR_CATEGORY_E

This object represents one of the error categories.

This topic lists the expressions that are provided by this class.

Enumerations

AAA_LOGIN_REDIRECT

This category of errors represents all return codes while

redirecting user for login.

ERR_AAA_ALLOC

This category identifies all memory allocation issue while

processing AAA traffic.

ERR_AAA_AUTHN

This category of errors identifies the errors AAA module

encounters while handling client authentication.

ERR_AAA_BAD_REQUEST

This error is set when AAA module sees a malformed message

ERR_AAA_C2C

This category of errors identifies the inter-node

communication failures that happen while processing AAA

traffic.

ERR_AAA_CONFIG

This category of errors identifies all configuration issues

while processing AAA traffic.

ERR_AAA_GENERIC

This category identifies the scenario when a AAA submodule sends a legacy error number

ERR_AAA_IIP

This category of errors identifies the issues while allocating intranet IP to a client.

ERR_AAA_KERBEROS

This category of errors represents all errors while processing Kerberos messages.

ERR_AAA_LOGIN_ATTEMPTS

This category of errors encompasses failures due to exceeding login attempts.

ERR_AAA_NOSERVICE

This category of errors holds all the cases where AAA module fails to establish backend server connection.

ERR_AAA_PAGE_NOT_FOUND

This error is set when a request for a non-existent page is seen by AAA module

ERR_AAA_POLICY

This category of errors identifies the failure in policy subsystem while evaluation of policies.

ERR_AAA_REASSEMBLY

This category of errors identifies the errors when AAA

module tries to accumulate the post data.

ERR_AAA_SAML

This category of errors represents all errors while

processing SAML (Secure Assertion Markup Language) messages.

ERR_AAA_UNAUTHORIZED

This error denotes a failed access to a resource

ERR_UNKNOWN

This category of errors is for all such errors for which

category is unknown.

Expressions

EQ(error_category_ee)

The result is Boolean TRUE if the input argument is equal to

the error presented by this object.

Parameters(expressions not allowed):

e -

```
Error enumeration
```

Returns: bool_at

NE(error_category_ee)

The result is Boolean TRUE if the input argument is not equal to

the error presented by this object.

Parameters(expressions not allowed):

e -

```
Error enumeration
```

Returns: bool_at

ERROR_T

This object represents Error encountered while processing the last

operation.

This topic lists the expressions that are provided by this class.

CATEGORY

Returns the error category.

Returns: error_category_e

ETHER_REQ_PROTOCOL_T

Provides access to MAC Address in Ethernet Requests.

This topic lists the expressions that are provided by this class.

DSTMAC

Returns MAC address present in the destination field of the Ethernet

header. All MAC addresses are given as colon-delimited hexadecimal

codes in the format ##:##:##:##:##:##. Valid values for

each # include numbers 0-9 and letters A through F.

Returns: mac_address_at

ETHERTYPE

Returns the EtherType of the current Ethernet frame.

Returns: ether_type_t

SRCMAC

Returns MAC address present in the source field of the Ethernet

header. All MAC addresses are given as colon-delimited hexadecimal

codes in the format ##:##:##:##:##:##. Valid values for

each # include numbers 0-9 and letters A through F.

Returns: mac_address_at

ETHER_RES_PROTOCOL_T

Provides access to MAC Address in Ethernet Response.

This topic lists the expressions that are provided by this class.

DSTMAC

Returns MAC address present in the destination field of the Ethernet

header

Returns: mac_address_at

ETHERTYPE

Returns the EtherType of the current Ethernet frame.

Returns: ether_type_t

SRCMAC

Returns MAC address present in the source field of the Ethernet

header

Returns: mac_address_at

ETHER_TYPE_E

This object represents one of the ether types.

This topic lists the expressions that are provided by this class.

Enumerations

ARP

ETHERTYPE_UNKNOWN

IPV4

IPV6

PUP

REVARP

VLAN

Expressions

EQ(ether_type_eethertype)

The result is Boolean TRUE if the input argument is equal to

the ethertype represented by this object.

Parameters(expressions not allowed):

ethertype -

```
EtherType
```

Returns: bool_at

NE(ether_type_eethertype)

The result is Boolean TRUE if the input argument is not equal to

the ethertype represented by this object.

Parameters(expressions not allowed):

ethertype -

```
EtherType
```

Returns: bool_at

ETHER_TYPE_T

This object provides Num operations and EtherType Enum

operations on the EtherType.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from ether_type_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(ether_type_eethertype)

The result is Boolean TRUE if the input argument is equal to

the ethertype represented by this object.

Parameters(expressions not allowed):

ethertype -

```
EtherType
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(ether_type_eethertype)

The result is Boolean TRUE if the input argument is not equal to

the ethertype represented by this object.

Parameters(expressions not allowed):

ethertype -

```
EtherType
```

Returns: bool_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert ether_type_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

HTTP_CACHE_CONTROL_T

This object provides for the ability to work on HTTP Cache-Control headers.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from http_header_t, nvlist_at.

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

> the number value to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

> the time value to append.

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

> the unsigned long value to append.

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

Return a Number Value giving the number of name-value

components in the list. If the last Name-Value List

component is of zero length or is all spaces then it is

ignored.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_INVALID</b></p><div><p>Returns TRUE or FALSE
depending on whether invalid
```

Cache-Control headers are present. See RFC 2616 for more details.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
IS_MAX_AGE

Returns TRUE or FALSE depending on whether the "max-age"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at
IS_MAX_STALE

Returns TRUE or FALSE depending on whether the "max-stale"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_MIN_FRESH

Returns TRUE or FALSE depending on whether the "min-fresh"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_MUST_REVALIDATE

Returns TRUE or FALSE depending on whether the "must-revalidate"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_NO_CACHE

Returns TRUE or FALSE depending on whether the "no-cache"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_NO_STORE

Returns TRUE or FALSE depending on whether the "no-store"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_NO_TRANSFORM

Returns TRUE or FALSE depending on whether the "no-transform"

Cache-Control header is present. However,

HTTP.REQ.CACHE_CONTROL.IS_NO_TRANSFORM will always return false

since the "no-transform" cache-control header can only be present

in the response. See RFC 2616 for more details.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>IS_ONLY_IF_CACHED</b></
p><div><p>Returns TRUE or FALSE depending on whether the "only-if-cached"
```

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_PRIVATE

Returns TRUE or FALSE depending on whether the "private"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_PROXY_REVALIDATE

Returns TRUE or FALSE depending on whether the "proxy-revalidate"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

IS_PUBLIC

Returns TRUE or FALSE depending on whether the "public"

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at

*** IS_STRINGMAP_KEY**(text_tstring_map_name)

Returns true if the string that it operates on is

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_S_MAXAGE</b></p><div><p>Returns TRUE or FALSE
depending on whether the "s-maxage"
```

Cache-Control header is present. See RFC 2616 for more details.

Returns: bool_at
*** IS_TIME_AT**

Results in Boolean TRUE if the Text follows any of the

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
IS_UNKNOWN

Returns TRUE or FALSE depending on whether unknown

Cache-Control headers are present. See RFC 2616 for more details.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
```

```
div><div><div><p><b>MAX_AGE</b></p><div><p>Returns the value of the HTTP
Cache-Control header "maxage".
```

If this header is absent or invalid, 0 is returned.

See RFC 2616 for more details.

Returns: num_at
MAX_STALE

Returns the value of the HTTP Cache-Control header "maxstale".

If this header is absent or invalid, 0 is returned.

See RFC 2616 for more details.

Returns: num_at
*** MAX**(text_ts)

Compares the string argument with the current string and

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>MIN_FRESH</b></p><div><p>Returns the value
of the HTTP Cache-Control header "min-fresh".
```

If this header is absent or invalid, 0 is returned.

See RFC 2616 for more details.

Returns: num_at
*** MIN**(text_ts)

Compares the string argument with the current string and

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAMES</b></p><div><p>Extracts a
list of only the names from the name-value pair list.
```

For example the list 'a=b,c="d=e,f",g=h' will result in the list

'a,c,g'. Empty elements will be ignored.

Returns: list_t
*** NAME**(uint i)

Operate as Text on the name component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(uint i)</
p><div><p>Returns the element in the list identified by the parameter.
```

The parameter is a 0-based index in to the list. The result

is a string. If the specified index is greater than the

number of components then a zero length string is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
```

```
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the name-
value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the name-value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>REPLACE_ALL_VALUES</b>(<a
href="text_t.html">text_t</a>value)</p><div><p>Replaces all the values in
each name-value pair with the specified
```

string argument.

For example, given argument string "?" the list 'a=b,c=d,e=f' will

result in the name-value list 'a=?,c=?,e=?'. Quotes will be stripped

from the values being replaced. Empty values will be replaced with

the value provided. Empty elements will be preserved.

Parameters(expressions not allowed):

value -

```
value to replace existing values</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```


use text.

Parameters(expressions not allowed):

charset -


```
Character Set</p></div><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```


many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>S_MAXAGE</b></p><div><p>Returns
the value of the HTTP Cache-Control header "s-maxage".
```

If this header is absent or invalid, 0 is returned.

See RFC 2616 for more details.

Returns: num_at
*** TO_LOWER**

Converts the selected text to lower case. For example, the string "ABCd" is converted to "abcd"

Returns: text_t
*** TO_UPPER**

Converts the selected text to upper case.

For example, the string "abcD" is converted to "ABCD"

Returns: text_t
*** TRUNCATE**(char c, uint i)

Select the Text after truncating a suffix from the current

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

TYPECAST_HTTP_HEADER_T

Convert http_cache_control_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : Typecast an HTTP Header object of a specific type to HTTP

Header.For example: HTTP.REQ.CONTENT_LENGTH.TYPECAST_HTTP_HEADER_T)

Returns: http_header_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result

ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at

*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUES**

Extracts a list of only the values from the name-value pair list.

For example the list 'a=b,c=d,e=f' will result in the list

'b,d,f'. Note that lists do not honor the quote characters, so

any nvlist_at entry that has quotes and a quoted entry includes a

separator will be split at that embedded separator. Quote

characters will not be stripped from the values. Empty elements

and empty values will be ignored.

Returns: list_t

*** VALUE**(uint i)

Operate as Text on the value component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the value
component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the value component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XML_DECRYPT</b>(<a href="text_t.html">text_t</a>certKeyName)</
p><div><p>Decrypt the selected content, which was encrypted using the W3C
XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{
```

```
              name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named
```

```
      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost',


        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
```

```
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton"</
p> Parameters(expressions not allowed): xpathex -

XPath Expression Returns: <a
href="text_t.html">text_t </body></html>

HTTP_CONTENT_LENGTH_T

The HTTP.REQ.CONTENT_LENGTH or HTTP.RES.CONTENT_LENGTH object can be

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from http_header_t, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An HTTP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: HTTP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 HTTP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
```

```
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_content_length_t.html">http_content_length_t</a></p></div></
div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="http_content_length_t.html">http_content_length_t</a></p></div></
div></div><div><div><p><b>* </b><b>SIGNED16_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
16-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as
DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
TYPECAST_HTTP_HEADER_T

Convert http_content_length_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : Typecast an HTTP Header object of a specific type to HTTP

Header.For example: HTTP.REQ.CONTENT_LENGTH.TYPECAST_HTTP_HEADER_T)

Returns: http_header_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work
```

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

```
produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

TYPECAST_NUM_AT

Convert http_content_length_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -
```

```
                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -
```

quote-character. Returns: nvlist_t * TYPECAST_RADIUS_AVP_T Convert text_t to radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_t : Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

destination type. (tcp_options_t : TCP Protocol Options) Returns:
tcp_options_t </
div> * TYPECAST_TCP_OPTIONS_TYPE_T</
p> Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p> Returns: <a
href="tcp_options_type_t.html">tcp_options_type_t </
div> * TYPECAST_TCP_REPEATER_OPTION_T</
p> Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater) Returns: <a
href="tcp_repeater_option_t.html">tcp_repeater_option_t </
div> * TYPECAST_TEXT_T Typecast
the Number Value to its string representation. Returns: <a
href="text_t.html">text_t * </
b>TYPECAST_TIME_AT Typecast the Number Value to a Time
value. Returns: time_at </
div> * TYPECAST_TIME_T</
p> Convert text_t to time_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An HTTP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>WEEKDAY_STRING</b></
p><div><p>Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,
```

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
```

```
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)
```

```
  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)



      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
           "person":{


             "name":'Milton'


           }


       },


       "title":'Paradise Lost',


       "price":"36"


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


     This will convert the string '36' string to double if it is


     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
```

```
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)
```

```
      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

HTTP_COOKIE_T

This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from http_header_t, nvlist_at.

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

> the IPv4 address to append.

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

> the number value to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

> the time value to append.

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

Return a Number Value giving the number of name-value components in the list. If the last Name-Value List component is of zero length or is all spaces then it is ignored.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a CRC (cyclic redundancy check) function to the selected text. This is the CRC function used by Ethernet, MPEG-2, PKZIP, GZIP, PNG, and many other applications. For example, the expression to compute the CRC value of the (first 1000 bytes of the) body of an HTTP request is: HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured text encoding methods like URLENCODED, BACKSLASH_ENCODED, PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and NO_PLUS_AS_SPACE. Text encoding methods are set using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"

    Percent: "%"

    Ampersand: ";"

    Plus: "+"

    Comma: ","

    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAMES</b></p><div><p>Extracts a
list of only the names from the name-value pair list.
```

For example the list 'a=b,c="d=e,f",g=h' will result in the list

'a,c,g'. Empty elements will be ignored.

Returns: list_t
*** NAME**(uint i)

Operate as Text on the name component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(uint i)</
p><div><p>Returns the element in the list identified by the parameter.
```

The parameter is a 0-based index in to the list. The result

is a string. If the specified index is greater than the

number of components then a zero length string is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the name-
value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the name-value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted cannot be longer than the public key length minus 11 bytes for padding. For example, for a 1024-bit (128 byte) key, the input text cannot be longer than 128 - 11 = 117 bytes. An Undef error will occur if the input text is too long. A null input will also cause an Undef error. Returns the (unencoded binary) encrypted text, which is typically base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +
```

```
"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>REPLACE_ALL_VALUES</b>(<a
href="text_t.html">text_t</a>value)</p><div><p>Replaces all the values in
each name-value pair with the specified
```

string argument.

For example, given argument string "?" the list 'a=b,c=d,e=f' will

result in the name-value list 'a=?,c=?,e=?'. Quotes will be stripped

from the values being replaced. Empty values will be replaced with

the value provided. Empty elements will be preserved.

Parameters(expressions not allowed):

value -

```
value to replace existing values</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_cookie_t.html">http_cookie_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_cookie_t.html">http_cookie_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```


Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```


treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

TYPECAST_HTTP_HEADER_T

Convert http_cookie_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : Typecast an HTTP Header object of a specific type to HTTP

Header.For example: HTTP.REQ.CONTENT_LENGTH.TYPECAST_HTTP_HEADER_T)

Returns: http_header_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result
```

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

ABCD:1234::ABCD5) Returns: ipv6_address_at * TYPECAST_IPV6_ADDRESS_T Convert text_t to ipv6_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the

```
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_header_t to text_t. Will raise an undef if the value is null, empty,
```

```
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
```

```
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id

- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
```

```
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUES**

Extracts a list of only the values from the name-value pair list.

For example the list 'a=b,c=d,e=f' will result in the list

'b,d,f'. Note that lists do not honor the quote characters, so

any nvlist_at entry that has quotes and a quoted entry includes a

separator will be split at that embedded separator. Quote

characters will not be stripped from the values. Empty elements

and empty values will be ignored.

Returns: list_t
*** VALUE**(uint i)

Operate as Text on the value component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the value
component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the value component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XML_DECRYPT</b>(<a href="text_t.html">text_t</a>certKeyName)</
```

```
p><div><p>Decrypt the selected content, which was encrypted using the W3C
XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
```

```
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)
```

```
  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)



      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
           "person":{


             "name":'Milton'


           }


       },


       "title":'Paradise Lost',


       "price":"36"


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


     This will convert the string '36' string to double if it is


     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
```

```
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)
```

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t </body></html>

HTTP_DATE_T

The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from http_header_t, time_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

time value.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value is between

the Lower and the Upper time arguments. Both the Lower and

the Upper time arguments must be fully specified.

For example -

GMT 1995 Jan is fully specified. But GMT Jan or GMT 1995 20

or GMT Jan Mon_2 are not fully specified. The Upper and

Lower arguments must be both GMT or both Local. The day of

the week must not be present in either the Lower or the

Upper argument. It is OK to specify the day of the month as

the 1st, 2nd, 3rd or 4th weekday of the month (example Wed_3

is the 3rd Wednesday of the month).

The Upper Time argument must be bigger than the Lower Time

argument.

Assume the current time value is GMT 2005 May 1

10h 15m 30s. This is the 1st Sunday of the month of May in 2005.

BETWEEN(GMT 2004, GMT 2006): TRUE BETWEEN(GMT 2004 Jan, GMT 2006 Nov): TRUE BETWEEN(GMT 2004 Jan, GMT 2006): TRUE BETWEEN(GMT 2005 May Sun_1, GMT 2005 May Sun_3): TRUE BETWEEN(GMT 2005 May 1, GMT May 2005 1): TRUE BETWEEN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
Lower Time
```

time2 -

```
Upper Time
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_attime1, time_attime2)

Returns Boolean TRUE if the time value is greater than or

equal to the Lower argument and the time value is lesser

than or equal to the Upper argument.

Parameters:

time1 -

```
Lower GMT Time Value
```

time2 -

```
Upper GMT Time Value
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An HTTP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: HTTP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 HTTP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DAY**

Extracts the day of the month from the current system time, and

returns that value as a number from 1 through 31 representing that

day.

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

EQ(GMT 2005): TRUE EQ(GMT 2005 Dec): FALSE EQ(Local 2005 May): TRUE or FALSE depending on the current

timezone. EQ(GMT 10h): TRUE EQ(GMT 10h 30s): TRUE EQ(GMT May 10h): TRUE EQ(GMT Sun): TRUE EQ(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

*** EQ**(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is greater
```

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

GE(GMT 2004): TRUE

GE(GMT 2005 Jan): TRUE

GE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

GE(GMT 8h): TRUE

GE(GMT 30m): FALSE

GE(GMT May 10h): TRUE

GE(GMT May 10h 0m): TRUE

GE(GMT Sun): TRUE

GE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

greater than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>GT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is greater
```

than the argument time.

Assume the current time value is GMT

2005 May 1 10h 15m 30s. This is the 1st Sunday of the month

of May in 2005.

GT(GMT 2004): TRUE

GT(GMT 2005 Jan): TRUE

GT(Local 2005 May): TRUE or FALSE depending on the current

timezone.

GT(GMT 8h): TRUE

GT(GMT 30m): FALSE

GT(GMT May 10h): FALSE

GT(GMT May 10h 0m): TRUE

GT(GMT Sun): FALSE

GT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

greater than the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>HASH</b></p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HOURS</b></p><div><p>Extracts the hour from the current system
time, and returns that
```

value as an integer from 0 to 23.

Returns: num_at
*** HTML_XML_SAFE**

Apply XML safe transform and select text.

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** INSTANCE**(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is lesser
```

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LE(GMT 2006): TRUE

LE(GMT 2005 Dec): TRUE

LE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

LE(GMT 8h): FALSE

LE(GMT 30m): TRUE

LE(GMT May 10h): TRUE

LE(GMT Jun 11h): TRUE

LE(GMT Wed): TRUE

LE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

less than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>LOCATION_LONG</b></p><div><p>Get the location qualifier from
CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is lesser
```

than the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LT(GMT 2006): TRUE

LT(GMT 2005 Dec): TRUE

LT(Local 2005 May): TRUE or FALSE depending on the current

timezone.

LT(GMT 8h): FALSE

LT(GMT 30m): TRUE

LT(GMT May 10h): FALSE

LT(GMT Jun 11h): TRUE

LT(GMT Wed): TRUE

LT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

less than the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MINUTES</b></p><div><p>Extracts the minute
from the current system time, and returns that
```

value as an integer from 0 to 59.

Returns: num_at
*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MONTH</b></p><div><p>Extracts the month from
the current system time, and returns that
```

value as an integer from 1 (January) to 12 (December).

Returns: num_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

not equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

NE(GMT 2005): FALSE

NE(GMT 2005 Dec): TRUE

NE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

NE(GMT 10h): FALSE

NE(GMT 10h 30s): FALSE

NE(GMT May 10h): FALSE

NE(GMT Sun): FALSE

NE(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

not equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>NORMALIZE_URL</b></p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>RELATIVE_BOOT</b></p><div><p>Calculates the number of seconds to the
closest previous or scheduled
```

reboot, and returns an integer. If the closest boot time is in the past,

the integer is negative; if it is in the future, the integer is positive.

Returns: num_at
*** RELATIVE_NOW**

Calculates the number of seconds between the current system time and

the specified time, and returns an integer showing that number. If the

designated time is in the past, the integer is negative; if it is in

the future, the integer is positive.

Returns: num_at
*** REPEAT**(uint i)

Produces a string which contains 'i' sequential copies of the input

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SECONDS</b></p><div><p>Extracts the seconds from the current system
time, and returns that
```

value as an integer from 0 to 59.

Returns: num_at

*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_date_t.html">http_date_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
```

```
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="http_date_t.html">http_date_t</a></p></div></
div><div><div><p><b>* </b><b>SIGNED16_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
16-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
```

```
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```


treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

TYPECAST_HTTP_HEADER_T

Convert http_date_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : Typecast an HTTP Header object of a specific type to HTTP

Header.For example: HTTP.REQ.CONTENT_LENGTH.TYPECAST_HTTP_HEADER_T)

Returns: http_header_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result
```

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

ABCD:1234::ABCD5) Returns: ipv6_address_at * TYPECAST_IPV6_ADDRESS_T Convert text_t to ipv6_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the

```
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
TYPECAST_NUM_AT

Convert http_date_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
```

```
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Typecast
the Time Value to its string representation.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>TYPECAST_TIME_AT</b></p><div><p>Convert http_date_t to
time_at. Will raise an undef if the value is null, empty, or the value has
```

```
an invalid format for the destination type. (time_at : Treats a number
value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```


the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

```
    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** VALUE**(uint lineno)

An HTTP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>WEEKDAY</b></
p><div><p>Results in a Number Value between 0 (Sunday) and 6
```

(Saturday) which gives the week day in the time value.

Returns: num_at
*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t
*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t
*** WITHIN**(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value lies

within all the ranges defined by the Lower and Upper time

arguments.

If an element of time like day or hour is left

unspecified in the Lower argument then it is assumed to have the

lowest value of its range.

If an element is left

unspecified in the Upper argument then it is assumed to have the

highest value of its range.

If the year is specified in one

of the argument then it must be specified in the other argument as

well.

The ranges for different elements of time are month

1-12, day 1-31, weekday 0-6, hour 0-23, minutes 0-59 and

seconds 0-59.

Each element of time defines a range. For the result to be

TRUE each element of time in the time value must lie in the

corresponding range present in the Lower and Upper

arguments.

Assume the time value is GMT 2005 May 10 10h 15m 30s. This

is the second Tuesday of the month.

WITHIN(GMT 2004, GMT 2006): TRUE

WITHIN(GMT 2004 Jan, GMT 2006 Mar): FALSE (May doesn't fall in

the Jan-Mar range.)

WITHIN(GMT Feb, GMT): TRUE (May falls in the Feb-Dec range.)

WITHIN(GMT Sun_1, GMT Sun_3): TRUE (2nd Tuesday lies within

1st Sunday and the 3rd Sunday.)

WITHIN(GMT 2005 May 1 10h, GMT May 2005 1 17h): TRUE

WITHIN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
                        Lower Time</p><p>time2 -



                        Upper Time</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XML_DECRYPT</b>(<a href="text_t.html">text_t</a>certKeyName)</
p><div><p>Decrypt the selected content, which was encrypted using the W3C
XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:
```

```
{


   "Book":{


      "creator":{


         "person":{


           "name":'Milton'


         }


      },


      "title":'Paradise Lost'


   }


}


HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)
```

This expression when operated on the above JSON file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

```
  creator:{

     person:{

       name:'Milton'

     }

  }
```

The original JSON document is not disturbed but the

part of the JSON body selected by the

```
      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'



           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.
```

```
      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)


      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"
```

```
      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)


      This will return true as there is a node called creator
```

```
      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>YEAR</b></p><div><p>Extracts the year from the current system time,
and returns that
```

value as a four-digit integer.

Returns: num_at

</body></html>

HTTP_HEADER_T

The HTTP.REQ.HEADER("header name") or HTTP.RES.HEADER("header name") object represents the multiple lines of a particular HTTP header type. This object is also a Text object. Direct Text operations on this object will mostly operate only on the value in the last header line of this type. Some Text operations like CONTAINS have been overridden to operate on values in all the header lines of this type.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***** symbol are inherited/promoted from text_t.

AFTER_REGEX(re regex)

The AFTER_REGEX operator in this object overrides the Text AFTER_REGEX operator to operate on all the instances of the current header type. Selects the Text after the text that matches the input regular expression in any instance of the header value. The order in which the header instances will be matched is from the last to the first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

AFTER_STR(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

BEFORE_REGEX(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

BEFORE_STR(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

CONTAINS_ANY(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

CONTAINS_INDEX(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

CONTAINS(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

COUNT

An HTTP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: HTTP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 HTTP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

EQUALS_ANY(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

EQUALS_INDEX(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

INSTANCE(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>REGEX_MATCH</b>(re regex)</
p><div><p>The REGEX_MATCH operator in this object overrides the Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT
operator in this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
```

```
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>The SUBSTR operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -


```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6
```

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

```
<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -
```

```
                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -
```

quote-character. Returns: nvlist_t * TYPECAST_RADIUS_AVP_T Convert text_t to radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_t : Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

destination type. (tcp_options_t : TCP Protocol Options) Returns:
tcp_options_t </
div> * TYPECAST_TCP_OPTIONS_TYPE_T</
p> Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p> Returns: <a
href="tcp_options_type_t.html">tcp_options_type_t </
div> * TYPECAST_TCP_REPEATER_OPTION_T</
p> Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater) Returns: <a
href="tcp_repeater_option_t.html">tcp_repeater_option_t </
div> TYPECAST_TEXT_T Convert
http_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.) Returns: <a
href="text_t.html">text_t * </
b>TYPECAST_TIME_AT Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from

beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
VALUE(uint lineno)

An HTTP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document

and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child
```

```
  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)


  returns the numeric value of the height attribute of the


  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:

  {

     "Book":{

        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is

    in the first 1000 bytes of the document.</p><p><b>Parameters</
```

```
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t </body></html>

HTTP_HOSTNAME_T

This object provides operations for parsing the HTTP hostname as it

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

DOMAIN

Operate as Text on the Domain name part of the hostname.

If the hostname is www.foobar.com or www.foobar.com:8080, then

domain is foobar.com. This operation will return incorrect

results if the hostname has an IP address. This method also sets

the text mode to case-insensitive. So all text operations after

this method will operate in a case-insensitive manner.

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

EQ(text_thostname)

Operates on the text contents of hostname. Results in Boolean TRUE

value if the hostname matches the input argument.

The comparison will be case-insensitive and if textmode is URLENCODED

the hostname will be decoded before comparison.

If hostname is www.foobar.com or www.foobar.com., then

EQ(www.foobar.com) returns TRUE.

Parameters:

hostname -

```
Hostname
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

Based on RFC2396, following are considered URL safe characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

```
Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at

*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

PORT

Operate as Number on the port part of the hostname. The

string following and including the first : is considered the

port value.

Examples:

If the hostname is www.foo.com:8080 then the port

is ":8080".

If the hostname is www.foo.com: then the port is

":".

If the hostname is www.foo.com then the port is "" and

points to just after ".com".

When the numerical value in the port is missing then it

assumes a default value of 80 or 443 (for HTTPS connections).

Returns: num_t

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SERVER</b></p><div><p>Operate as Text on the Server
name part of the hostname.
```

If the hostname is www.foobar.com or www.foobar.com:8080, then

server is www.foobar.com. This method also sets the text mode to

case-insensitive. So all text operations after this method

will operate in a case-insensitive manner.

Returns: text_t
*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_hostname_t.html">http_hostname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_hostname_t.html">http_hostname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
```

```
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_hostname_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'


          }


       },


       "title":'Paradise Lost',


       "price":"36"


    }


 }


 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


     This will convert the string '36' string to double if it is


     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"

      in the first 1000 bytes of the body. Since this is a node-set

      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

HTTP_METHOD_E

This object represents one of the HTTP Request Methods.

This topic lists the expressions that are provided by this class.

Enumerations

CONNECT

DELETE

GET

HEAD

OPTIONS

POST

PUT

TRACE

UNKNOWN_METHOD

Expressions

EQ(http_method_em)

The result is Boolean TRUE if the input argument is equal to

the HTTP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
Http Request Method Name
```

Returns: bool_at

HTTP_METHOD_T

This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, http_method_e.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(http_method_em)

The result is Boolean TRUE if the input argument is equal to

the HTTP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
Http Request Method Name
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
    Percent: "%"

    Ampersand: ";"

    Plus: "+"

    Comma: ","

    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'

- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AUnw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_method_t.html">http_method_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_method_t.html">http_method_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
```

```
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_method_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"


      in the first 1000 bytes of the body. Since this is a node-set


      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

HTTP_PROTOCOL_T

Models the HTTP message types.

This topic lists the expressions that are provided by this class.

REQ

Operates on HTTP Request.

Returns: http_req_t

RES

Operates on HTTP Response.

Returns: http_res_t

HTTP_REQ_T

Operates on HTTP Request.

This topic lists the expressions that are provided by this class.

BODY

Returns the body of the HTTP request as a multiline text object. NOTE: This usage is deprecated;

you should use HTTP.REQ.BODY(uint) instead.

Returns: text_t

BODY(uint i)

Operate as Text on the HTTP response body. The length of the

body is equal to the number in the argument. If there aren't

enough characters in the body then the entire body is

selected.

Parameters:

i -

```
Body length
```

Returns: text_t

CACHE_CONTROL

Returns the HTTP Cache-Control object.

Returns: http_cache_control_t

CONTENT_LENGTH

Returns the length of the HTTP request as a number.

Returns: http_content_length_t

COOKIE

(Name/Value List) Returns the contents of the HTTP Cookie header as a name/value list.

Returns: http_cookie_t

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

DATE

Returns the contents of the HTTP Date header as text or as a date object.The date formats

recognized are:

RFC822. Sun, 06 Jan 1980 08:49:37 GMT

RFC850. Sunday, 06-Jan-80 9:49:37 GMT

ASCTIME. Sun Jan 6 08:49:37 1980

Returns: http_date_t

FULL_HEADER

Returns the entire request header (including the

terminating \r\n\r\n)

Returns: text_t

HEADER(text_theader_name)

Returns the contents of the HTTP header specified by the String argument.

The header name cannot be longer than 32 characters.

Parameters(expressions not allowed):

header_name -

```
Header Name
```

Returns: http_header_t

HOSTNAME

Obtain the HTTP Host Name object from this request. If the target hostname is present in the first line of the request then that is selected. Otherwise the value in the last occurrence of the HOST header is selected.

Returns: http_hostname_t

IS_NTLM_OR_NEGOTIATE

Returns TRUE if request is part of NTLM or NEGOTIATE connection

Returns: bool_at

IS_VALID

Returns TRUE if the HTTP request is properly formed.

Returns: bool_at

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

METHOD

Returns the HTTP method.

Returns: http_method_t

TRACKING

Returns the HTTP body tracking mechanism.

Returns: http_tracking_e

TXID

Returns the HTTP Transaction ID. The value is a function of

an internal transaction number, system boot time and system

mac address. HTTP.REQ.TXID is same as HTTP.RES.TXID.

Returns: text_t

URL

Obtain the HTTP URL object from the request. The Text object

backing the HTTP URL object has the Text Mode set to

URLENCODED by default.

Returns: http_url_t

USER

Returns the AAA User associated with the current HTTP transaction.

Returns: aaa_user_t

VERSION

Returns the HTTP version.

Returns: http_version_t

HTTP_RES_T

Operates HTTP Response.

This topic lists the expressions that are provided by this class.

BODY(uint i)

Operate as Text on the HTTP response body. The length of the body is equal

to the number in the argument. If there aren't enough characters in the

body then the entire body is selected.

Parameters:

i -

```
Body length
```

Returns: text_t

CACHE_CONTROL

Returns the HTTP Cache-Control object

Returns: http_cache_control_t

CONTENT_LENGTH

Returns the length of the HTTP response as a number.

Returns: http_content_length_t

DATE

Returns the contents of the HTTP Date header as text or as a date object.The date formats

recognized are:

RFC822. Sun, 06 Jan 1980 08:49:37 GMT

RFC850. Sunday, 06-Jan-80 9:49:37 GMT

ASCTIME. Sun Jan 6 08:49:37 1980

Returns: http_date_t

FULL_HEADER

Returns the entire response header (including the

terminating \r\n\r\n)

Returns: text_t

HEADER(text_theader_name)

Returns the contents of the HTTP headerspecified by the String argument.

The header name cannot be longer than 32 characters.

Parameters(expressions not allowed):

header_name -

```
Header Name
```

Returns: http_header_t

IS_CLIENT_ERROR

Returns True if response code is 4xx.

Returns: bool_at

IS_INFORMATIONAL

Returns True if response code is 1xx.

Returns: bool_at

IS_REDIRECT

Returns True if response code is 3xx.

Returns: bool_at

IS_SERVER_ERROR

Returns True if response code is 5xx.

Returns: bool_at

IS_SUCCESSFUL

Returns True if response code is 2xx.

Returns: bool_at

IS_VALID

Returns TRUE if the HTTP request is properly formed.

Returns: bool_at

MEDIA

Designates an expression that refers to media analytics

Returns: media_t

SET_COOKIE

Obtain the HTTP Set-Cookie Header Object from this response.

Returns: http_set_cookie_t

SET_COOKIE2

Obtain the HTTP Set-Cookie2 Header Object from this response.

Returns: http_set_cookie_t

STATUS

Returns the HTTP response status code.

Returns: num_t

STATUS_MSG

Returns the HTTP response status message.

Returns: text_t

TRACKING

Returns the HTTP body tracking mechanism.

Returns: http_tracking_e

TXID

Returns the HTTP Transaction ID. The value is a function of

an internal transaction number, system boot time and system

mac address. HTTP.REQ.TXID is same as HTTP.RES.TXID.

Returns: text_t

VERSION

Returns the HTTP version.

Returns: http_version_t

HTTP_SET_COOKIE_T

HTTP.RES.SET_COOKIE provides operations for parsing and operating on the SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is selected.

HTTP Set-Cookie header also supports special operations to extract cookies from the response. It is assumed that there will be only one cookie in a Set-Cookie header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***** symbol are inherited/promoted from http_header_t.

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text AFTER_REGEX operator to operate on all the instances of the current header type. Selects the Text after the text that matches the input regular expression in any instance of the header value. The order in which the header instances will be matched is from the last to the first.

For example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
HTTP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

HTTP/1.1 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
HTTP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / HTTP/1.1\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: HTTP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
HTTP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. HTTP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. HTTP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

COOKIE(text_tname)

Returns the Cookie with Cookie_name if present else NULL

(Text of length 0 pointing nowhere) is returned.

Raises UNDEF if more than 15 Set-Cookie headers are present

and the Cookie could not be found in the first 15 headers.

Parameters(expressions not allowed):

name -

```
Cookie_Name
```

Returns: cookie_t

COOKIE(text_tname, uint i)

Returns the i-th instance of Cookie with name if present

else NULL (Text with length 0 pointing nowhere) is returned.

The index of the Cookie component given by the argument is

0-based.

Raises UNDEF if more than 15 Set-Cookie headers are present

and the Cookie could not be found in the first 15 headers.

Parameters(expressions not allowed):

name -

```
Cookie_Name
```

i -

```
Index
```

Returns: cookie_t

*** COUNT**

An HTTP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

HTTP/1.1 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: HTTP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 HTTP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

EXISTS(text_tname)

Returns TRUE if the Cookie with name exists in the Set-Cookie header

else FALSE.

Raises UNDEF if more than 15 Set-Cookie headers are present

and the Cookie could not be found in the first 15 headers.

Parameters(expressions not allowed):

name -

```
Cookie_Name
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"


    Open and close parentheses: "(", ")"
```

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** INSTANCE**(uint lineno)

An HTTP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'

- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AUnw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_set_cookie_t.html">http_set_cookie_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_set_cookie_t.html">http_set_cookie_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

HTTP/1.1 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```


treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

TYPECAST_HTTP_HEADER_T

Convert http_set_cookie_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : Typecast an HTTP Header object of a specific type to HTTP

Header.For example: HTTP.REQ.CONTENT_LENGTH.TYPECAST_HTTP_HEADER_T)

Returns: http_header_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result
```

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

ABCD:1234::ABCD5) Returns: ipv6_address_at * TYPECAST_IPV6_ADDRESS_T Convert text_t to ipv6_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the

```
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_header_t to text_t. Will raise an undef if the value is null, empty,
```

```
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
```

```
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id

- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
```

```
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An HTTP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

HTTP/1.1 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: HTTP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

its string value is returned which is: "Milton"</
p> Parameters(expressions not allowed): xpathex -

XPath Expression Returns: <a
href="text_t.html">text_t </body></html>

HTTP_TRACKING_E

Provides the tracking type of the HTTP Request or Response. (Length, Fin, Chunk)

This topic lists the expressions that are provided by this class.

Enumerations

CHUNK

FIN

LENGTH

UNKNOWN_TRACKING

Expressions

EQ(http_tracking_et)

The result is Boolean TRUE if the tracking type of the HTTP

request or response matches the input argument.

Parameters(expressions not allowed):

t -

```
Http Tracking Mechanism
```

Returns: bool_at

HTTP_URL_T

HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header. The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

APPEND_QUERY_PARAMETER(text_tname, text_tvalue)

Returns the URL string with a query parameter

appended. This function appends query parameter

after Converting URL unsafe characters into

corresponding '%xx' values and retains the rest.

Parameters(expressions not allowed):

name -

```
query parameter name
```

value -

```
query parameter value
```

Returns: http_url_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>HOSTNAME</b></p><div><p>Operate as HTTP Host Name on
the hostname present in the
```

URL.

It can not be used in bi-directional policies.

Returns: http_hostname_t
*** HTML_XML_SAFE**

Apply XML safe transform and select text.

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"


    Open and close parentheses: "(", ")"
```

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
PATH

Operate as a / separated List on the path component of the

URL.

For example: If the URL is http://www.foo.com/a/b/c/bar.html?a=1 then

operation will select /a/b/c/bar.html. HTTP.REQ.URL.PATH.GET(1)

in the above example will select "a".

Returns: list_t
PATH_AND_QUERY

Operate as Text on the portion of the URL following the

hostname.

For example: If the URL is http://www.foo.com/a/b/c/bar.html?a=1 then

it will return /a/b/c/bar.html?a=1.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AUnw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>PROTOCOL</b></p><div><p>Operate as
Text on the protocol present in the URL.
```

It can not be used in bi-directional policies.

Returns: text_t
QUERY

Operate as a Name-Value List (with delimiters = and ;) on the

query component of the URL.

Returns: nvlist_t
*** REGEX_MATCH**(re regex)

Returns TRUE if and only if the text contains at least one match

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_url_t.html">http_url_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_url_t.html">http_url_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>SUFFIX</b></p><div><p>Operate as Text on
the filename suffix present in the
```

filename component of the URL.

For example: If the PATH is /a/b/c/foo.bar.html

then this operation will select html.

Returns: text_t
*** SUFFIX**(char c, uint i)

Select a suffix of the Text. The selected suffix is the

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -
```

```
                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.
```

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

```
ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at

*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
```

```
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
```

```
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_url_t to text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
```

```
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)
```

```
  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"
```

```
      in the first 1000 bytes of the body which is


        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is
```

```
      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:



  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'
```

```
            }

        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)


      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

HTTP_VERSION_T

HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>MAJOR</b></p><div><p>Operate as Number
on the major component of the HTTP Version
```

string.

Returns: num_t
*** MAP_STRING_DEFAULT_TO_KEY**(text_tstring_map_name)

Returns the value corresponding to the key in the

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>MINOR</b></p><div><p>Operate as Number on
the minor component of the HTTP Version
```

string.

Returns: num_t
*** MIN**(text_ts)

Compares the string argument with the current string and

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="http_version_t.html">http_version_t</a></p></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="http_version_t.html">http_version_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
http_version_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
  identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'



           }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

ICA_PROTOCOL_T

Models the ICA protocol.

This topic lists the expressions that are provided by this class.

SERVER

Operates on the ICA protocol data associated with the current packet.

Returns: ica_server_t

ICA_SERVER_T

Operates on the ICA flow.

This topic lists the expressions that are provided by this class.

IP

Returns the IP of the backend server.

Returns: ip_address_at

IPV6

Returns IPv6 Address of the backend server.

Returns: ipv6_address_at

PORT

Returns the destination port.

Returns: num_at

INTERFACE_REQ_T

The physical interface request

This topic lists the expressions that are provided by this class.

ID

Extracts the interface ID of the interface through which the current packet entered the Application Switch, and returns that ID.

Returns: interface_t

RXTHROUGHPUT

Returns an integer representing the raw received traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

RXTXTHROUGHPUT

Returns an integer representing the raw received and transmitted traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

TXTHROUGHPUT

Returns an integer representing the raw transmitted traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

INTERFACE_RES_T

The physical interface response

This topic lists the expressions that are provided by this class.

ID

Extracts the interface ID of the interface through which the current packet entered the Application Switch, and returns that ID.

Returns: interface_t

RXTHROUGHPUT

Returns an integer representing the raw received traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

RXTXTHROUGHPUT

Returns an integer representing the raw received and transmitted traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

TXTHROUGHPUT

Returns an integer representing the raw transmitted traffic throughput in kilobytes per second for the previous seven seconds.

Returns: num_at

INTERFACE_T

Network Interface ID

This topic lists the expressions that are provided by this class.

EQ(text_tid)

Returns Boolean TRUE if the interface's ID matches the argument.

The input string cannot be bigger than 9 characters. If Cluster is enabled then it will support 3-tuple interface ID too .Examples: CLIENT.INTERFACE.ID.EQ("1/1")

SERVER.INTERFACE.ID.EQ("LA/1")

CLIENT.INTERFACE.ID.EQ("0/1/1")

SERVER.INTERFACE.ID.EQ("1/LA/1")

CONNECTION.INTF.EQ("1/1")

Parameters(expressions not allowed):

id -

```
Interface ID
```

Returns: bool_at

NE(text_tid)

Returns Boolean TRUE if the interface's ID doesn't match the argument.

The input string cannot be bigger than 9 characters. If Cluster is enabled then it will support 3-tuple interface ID too .Examples: CLIENT.INTERFACE.ID.NE("1/1")

SERVER.INTERFACE.ID.NE("LA/1")

CLIENT.INTERFACE.ID.NE("0/1/1")

SERVER.INTERFACE.ID.NE("1/LA/1")

CONNECTION.INTF.NE("LO/1")

Parameters(expressions not allowed):

id -

```
Interface ID
```

Returns: bool_at

IPV6_ADDRESS_AT

Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions: IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.
```

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

```
Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".
```

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

```
ABCD:1234::ABCD5
```

This topic lists the expressions that are provided by this class.

APPEND(text_tstr)

Appends the given string to the string representation of the current

IPv6 address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

APPLY_MASK(ipv6_address_ati)

Returns IPV6 Address Value after applying subnet mask

which can take values between 0.0.0.0.0.0.0.0 and

ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: ipv6_address_at

EQ(ipv6_address_ati)

Returns Boolean TRUE if the IP address value is same as the

argument.

IPv6 Address specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

GET1

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 1000, the first set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET1.EQ(0x2058) will return true.

Returns: num_at

GET2

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 1001, the second set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET2.EQ(0x2) will return true.

Returns: num_at

GET3

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return CD10, the third set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET3.EQ(0x0) will return true.

Returns: num_at

GET4

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 0000, the fourth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET4.EQ(0x0) will return true.

Returns: num_at

GET5

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 0000, the fifth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET5.EQ(0x0) will return true.

Returns: num_at

GET6

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 89AB, the sixth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET6.EQ(0x0) will return true.

Returns: num_at

GET7

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 4567, the seventh set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET7.EQ(0x230) will return true.

Returns: num_at

GET8

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return CDEF, the eighth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET8.EQ(0x3a5a) will return true.

Returns: num_at

IN_SUBNET(ipv6_subnet_ats)

Returns Boolean TRUE if the IP address value is in the

subnet specified by the argument.

IPv6 Subnet specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF/60).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

s -

```
IP Subnet
```

Returns: bool_at

IS_IPV4

Returns TRUE or FALSE depending on whether this is an IPv4 client.

Returns: bool_at

LAT_LONG

Results in text_t containing location as cordinates.

Location will be of the form:

Latitude,Longitude Expression example: CLIENT.IPV6.SRC.LAT_LONG

For example: 36,-119

Returns: text_t

LOCATION

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 Expression example: CLIENT.IPV6.SRC.LOCATION

For example: NorthAmerica.US....

Returns: list_t

LOCATION(num_atn)

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 Expression example: CLIENT.IPV6.SRC.LOCATION(2)

For example: NorthAmerica.US

Parameters(expressions not allowed):

n -

```
is the level up to which location will be
```

returned, the first qualifier is numbered 1.

Returns: list_t

MATCHES_LOCATION(text_tlocation)

Results in Boolean TRUE value if and only if

the location of the IP matches the location argument.

Location string will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 Expression example:

CLIENT.IPV6.SRC.MATCHES_LOCATION("qual1.qual2.qual3...*")

For example:

CLIENT.IPV6.SRC.MATCHES_LOCATION("NorthAmerica.US.California...*")

Parameters(expressions not allowed):

location -

```
Location to be matched.
```

Returns: bool_at

MATCHES(text_thostname)

Returns Boolean TRUE if hostname IPV6 address matches the current IPV6

address.

The hostname string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

hostname -

```
HostName
```

Returns: bool_at

NE(ipv6_address_ati)

Returns Boolean TRUE if the IP address value is not same as the

argument.

IPv6 Address specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

SUBNET(num_atn)

Returns IPV6 Address Value after applying the subnet which can

take values between 0 and 128. For Example: CLIENT.IPV6.SRC.SUBNET(24)

Parameters(expressions not allowed):

n -

```
Subnet value. It can take values between 0 and 128.
```

Returns: ipv6_address_at

TYPECAST_TEXT_T

Typecast the IPv6 Address Value to its string representation.

Returns: text_t

IPV6_ADDRESS_PAIR_T

Provides operations for IPv6 address pair

This topic lists the expressions that are provided by this class.

EQ(ipv6_address_ati)

Returns boolean TRUE if either source IP or destination IP is same as the

argument. Example : CONNECTION.IPv6.EQ(2001:db8:0:0:1::1)

Parameters(expressions not allowed):

i -

```
IPv6 address
```

Returns: bool_at

NE(ipv6_address_ati)

Returns boolesn TRUE is neither source IP nor Destination IP is same as the

argument. Example : CONNECTION.IPv6.NE(2001:db8:0:0:1::1)

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

IPV6_ADDRESS_T

This object represents an IPv6 address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, ipv6_address_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

IPv6 address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPLY_MASK**(ipv6_address_ati)

Returns IPV6 Address Value after applying subnet mask

which can take values between 0.0.0.0.0.0.0.0 and

ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: ipv6_address_at

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(ipv6_address_ati)

Returns Boolean TRUE if the IP address value is same as the

argument.

IPv6 Address specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET1**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 1000, the first set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET1.EQ(0x2058) will return true.

Returns: num_at

*** GET2**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 1001, the second set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET2.EQ(0x2) will return true.

Returns: num_at

*** GET3**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return CD10, the third set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET3.EQ(0x0) will return true.

Returns: num_at

*** GET4**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 0000, the fourth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET4.EQ(0x0) will return true.

Returns: num_at

*** GET5**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 0000, the fifth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET5.EQ(0x0) will return true.

Returns: num_at

*** GET6**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 89AB, the sixth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET6.EQ(0x0) will return true.

Returns: num_at

*** GET7**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return 4567, the seventh set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET7.EQ(0x230) will return true.

Returns: num_at

*** GET8**

If the IP address in the current context is

1000:1001:CD10:0000:0000:89AB:4567:CDEF

then it will return CDEF, the eighth set of 128 bits of the IP

address. The IPv6 address is printed using hexadecimal digits.

If the Client's IPv6 address is 2058:2::230:3a5a then

CLIENT.IPV6.SRC.GET8.EQ(0x3a5a) will return true.

Returns: num_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IN_SUBNET**(ipv6_subnet_ats)

Returns Boolean TRUE if the IP address value is in the

subnet specified by the argument.

IPv6 Subnet specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF/60).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

s -

```
IP Subnet</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV4</b></p><div><p>Returns TRUE or FALSE
depending on whether this is an IPv4 client.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
```

```
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LAT_LONG**

Results in text_t containing location as cordinates.

Location will be of the form:

Latitude,Longitude

Expression example: CLIENT.IPV6.SRC.LAT_LONG

For example: 36,-119

Returns: text_t
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION</b></
p><div><p>Results in list_t containing location.
```

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

Expression example: CLIENT.IPV6.SRC.LOCATION

For example: NorthAmerica.US....

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LOCATION**(num_atn)

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

Expression example: CLIENT.IPV6.SRC.LOCATION(2)

For example: NorthAmerica.US

Parameters(expressions not allowed):

n -

```
is the level up to which location will be
```

returned, the first qualifier is numbered 1.
Returns: list_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MATCHES_LOCATION</b>(<a href="text_t.html">text_t</a>location)</
p><div><p>Results in Boolean TRUE value if and only if
```

the location of the IP matches the location argument.

Location string will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

Expression example:

CLIENT.IPV6.SRC.MATCHES_LOCATION("qual1.qual2.qual3...*")

For example:

CLIENT.IPV6.SRC.MATCHES_LOCATION("NorthAmerica.US.California...*")

Parameters(expressions not allowed):

location -

```
Location to be matched.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>MATCHES</b>(<a href="text_t.html">text_t</a>hostname)</
p><div><p>Returns Boolean TRUE if hostname IPV6 address matches the
current IPV6
```

address.

The hostname string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

hostname -

```
HostName</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ipv6_address_at.html">ipv6_address_at</a>i)</p><div><p>Returns
Boolean TRUE if the IP address value is not same as the
```

argument.

IPv6 Address specified in one of the valid IPv6 forms

like CLIENT.IPV6.DST.EQ(1000:1001:CD10:0000:0000:89AB:4567:CDEF).

All valid forms of specifying an IPv6 address are supported.

Parameters(expressions not allowed):

i -

```
IP Address</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
```

```
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="ipv6_address_t.html">ipv6_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

ascii value of character Y in hexadecimal.

For example, the encoded text "http%3a//" will be decoded to "http://"

2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the

ascii value of character Y in hexadecimal.

For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="ipv6_address_t.html">ipv6_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBNET**(num_atn)

Returns IPV6 Address Value after applying the subnet which can

take values between 0 and 128.

For Example: CLIENT.IPV6.SRC.SUBNET(24)

Parameters(expressions not allowed):

n -

```
Subnet value. It can take values between 0 and 128.</p></
div><p><b>Returns: </b><a href="ipv6_address_at.html">ipv6_address_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUBSTR_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Select any one sub-
string that matches any string
```

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
TYPECAST_IPV6_ADDRESS_AT

Convert ipv6_address_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

Format Returns: num_t * TYPECAST_NUM_T(num_format_eformat, num_atdefault_value) Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

Format default_value -

Number Returns: num_t * TYPECAST_NVLIST_T(char sep1, char sep2) Convert text_t to nvlist_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (nvlist_t : Two character delimiters have to be provided when

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
ipv6_address_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```


String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
  identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

IPV6_REQ_PROTOCOL_T

IPv6 Protocol Request.

This topic lists the expressions that are provided by this class.

DST

Returns IPv6 Address Value present in the destination field of the

IP Header.

Returns: ipv6_address_at

PROTOCOL

Returns the Protocol of the current packet.

Returns: ip_protocol_type_t

SRC

Returns IPv6 Address Value present in the source field of the

IP Header.

Returns: ipv6_address_at

IPV6_RES_PROTOCOL_T

IPv6 Protocol Response.

This topic lists the expressions that are provided by this class.

DST

Returns IPv6 Address Value present in the destination field of the

IP Header.

Returns: ipv6_address_at

PROTOCOL

Returns the Protocol of the current packet.

Returns: ip_protocol_type_t

SRC

Returns IPv6 Address Value present in the source field of the

IP Header.

Returns: ipv6_address_at

IPV6_SUBNET_AT

Provides operations for the integral IPv6 subnet address value.

This topic lists the expressions that are provided by this class.

APPEND(text_tstr)

Appends the given string to the string representation of the current

IPv6 subnet address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

EQ(ipv6_subnet_ati)

Returns Boolean TRUE if the IPv6 subnet address value is same as the

argument.

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64

Parameters:

i -

```
IP Subnet Address
```

Returns: bool_at

GET1

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 1000, the first set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET1.EQ(0x1000) will return true.

Returns: num_at

GET2

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 1001, the second set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET2.EQ(0x1001) will return true.

Returns: num_at

GET3

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return CD10, the third set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET3.EQ(0xCD10) will return true.

Returns: num_at

GET4

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 0000, the fourth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET4.EQ(0x0) will return true.

Returns: num_at

GET5

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 0010, the fifth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET5.EQ(0x0010) will return true.

Returns: num_at

GET6

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:00100:89AB:4567:CDEF/64

then it will return 89AB, the sixth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET6.EQ(0x89AB) will return true.

Returns: num_at

GET7

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 4567, the seventh set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET7.EQ(0x4567) will return true.

Returns: num_at

GET8

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return CDEF, the eigth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET8.EQ(0xCDEF) will return true.

Returns: num_at

NE(ipv6_subnet_ati)

Returns Boolean TRUE if the IPv6 subnet address value is NOT same as the

argument.

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64

Parameters:

i -

```
IP Subnet Address
```

Returns: bool_at

PREFIXLENGTH

Returns Network Perfix Length from the given IPv6 Network Address

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

Then, this function will return 64

Returns: num_at

TYPECAST_TEXT_T

Typecast the IPv6 Subnet Address Value to its string representation.

Returns: text_t

IPV6_SUBNET_T

This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, ipv6_subnet_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

IPv6 subnet address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(ipv6_subnet_ati)

Returns Boolean TRUE if the IPv6 subnet address value is same as the

argument.

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64

Parameters:

i -

```
IP Subnet Address
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET1**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 1000, the first set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET1.EQ(0x1000) will return true.

Returns: num_at

*** GET2**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 1001, the second set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET2.EQ(0x1001) will return true.

Returns: num_at

*** GET3**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return CD10, the third set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET3.EQ(0xCD10) will return true.

Returns: num_at

*** GET4**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 0000, the fourth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET4.EQ(0x0) will return true.

Returns: num_at

*** GET5**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 0010, the fifth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET5.EQ(0x0010) will return true.

Returns: num_at

*** GET6**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:00100:89AB:4567:CDEF/64

then it will return 89AB, the sixth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET6.EQ(0x89AB) will return true.

Returns: num_at

*** GET7**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return 4567, the seventh set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET7.EQ(0x4567) will return true.

Returns: num_at

*** GET8**

If the IPv6 subnet address in the current context is

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

then it will return CDEF, the eigth set of 16 bits in the 128 bit

address. The IPv6 address is printed using hexadecimal digits.

If the IPv6 address is 1000:1001:CD10:0000:0010:89AB:4567:CDEF/64 then

RADIUS.REQ.AVP(97).VALUE.TYPECAST_IPV6_SUBNET_AT.GET8.EQ(0xCDEF) will return true.

Returns: num_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="ipv6_subnet_at.html">ipv6_subnet_at</a>i)</p><div><p>Returns Boolean
TRUE if the IPv6 subnet address value is NOT same as the
```

argument.

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64

Parameters:

i -

```
IP Subnet Address</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIXLENGTH**

Returns Network Perfix Length from the given IPv6 Network Address

IPv6 Subnet Address specified must be in following form:

1000:1001:CD10:0000:0010:89AB:4567:CDEF/64

Then, this function will return 64

Returns: num_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -
```

```
                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="ipv6_subnet_t.html">ipv6_subnet_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http%3a//" will be decoded to "http://"
2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will

both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"

3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="ipv6_subnet_t.html">ipv6_subnet_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as
DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6
```

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

```
<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
TYPECAST_IPV6_SUBNET_AT

Convert ipv6_subnet_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -
```

```
                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -
```

quote-character. Returns: nvlist_t * TYPECAST_RADIUS_AVP_T Convert text_t to radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_t : Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

destination type. (tcp_options_t : TCP Protocol Options) Returns:
tcp_options_t </
div> * TYPECAST_TCP_OPTIONS_TYPE_T</
p> Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p> Returns: <a
href="tcp_options_type_t.html">tcp_options_type_t </
div> * TYPECAST_TCP_REPEATER_OPTION_T</
p> Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater) Returns: <a
href="tcp_repeater_option_t.html">tcp_repeater_option_t </
div> TYPECAST_TEXT_T Convert
ipv6_subnet_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.) Returns: <a
href="text_t.html">text_t * </
b>TYPECAST_TIME_AT Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```


supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```


String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from

beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;

- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                      XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by
```

```
      /Book/creator"

      in the first 1000 bytes of the body which is

        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost'
```

```
    }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)
```

```
      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.

    Since this is a node-set the corresponding string

    value will be returned which is "MiltonParadise
```

```
      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is
```

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

For the XML file:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)

This will return true as there is a node called creator

under the node Book with in the first 1000 bytes of the

document. Parameters(expressions not allowed):

```
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

IP_ADDRESS_AT

Provides operations for the 32-bit integral IP address value.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

IP address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

APPLY_MASK(ip_address_ati)

Returns IP Address Value after applying subnet mask which

can take values between 0.0.0.0 and 255.255.255.255.

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: ip_address_at

APPLY_MASK(ip_address_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

APPLY_MASK(num_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

APPLY_MASK(time_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

APPLY_MASK(unsigned_long_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

EQ(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(ip_address_ati)

Returns Boolean TRUE if the IP address value is same as the

argument. For example: CLIENT.IP.DST.EQ(10.100.10.100)

Parameters:

i -

```
IP Address
```

Returns: bool_at

EQ(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GET1

Returns a Number Value 10 if the IP address value is 10.100.200.1

Returns: num_at

GET2

Returns a Number Value 100 if the IP address value is 10.100.200.1

Returns: num_at

GET3

Returns a Number Value 200 if the IP address value is 10.100.200.1

Returns: num_at

GET4

Returns a Number Value 1 if the IP address value is 10.100.200.1

Returns: num_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

IN_SUBNET(ip_subnet s)

Returns Boolean TRUE if the IP address value is in the

subnet specified by the argument. For example: CLIENT.IP.DST.IN_SUBNET(10.100.10.100/18)

Parameters(expressions not allowed):

s -

```
IP Subnet
```

Returns: bool_at

IPREP_IS_MALICIOUS

IP Reputation.

Returns TRUE if the client IP is categorized as malicious

else returns FALSE.

Returns: bool_at

IPREP_THREAT_CATEGORY(ip_protocol_iprep_category_ecategory)

IP Reputation.

Returns TRUE if the client IP is categorized as malicious

and threat category matches category argument.

else returns FALSE.

Parameters(expressions not allowed):

category -

```
is IP category
```

Returns: bool_at

IS_IPV6

Returns TRUE or FALSE depending on whether this is an IPv6 client.

Returns: bool_at

LAT_LONG

Results in text_t containing location as cordinates.

Location will be of the form:

Latitude,Longitude Expression example: CLIENT.IP.SRC.LAT_LONG

For example: 36,-119

Returns: text_t

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LOCATION

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 Expression example: CLIENT.IP.SRC.LOCATION

For example: NorthAmerica.US....

Returns: list_t

LOCATION(num_atn)

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 Expression example: CLIENT.IP.SRC.LOCATION(2)

For example: NorthAmerica.US

Parameters(expressions not allowed):

n -

```
is the level up to which location will be
```

returned, the first qualifier is numbered 1.

Returns: list_t

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

MATCHES_LOCATION(text_tlocation)

Results in Boolean TRUE

value if the location of the IP matches the location argument.

Location string can be of the form:

qual1.qual2.qual3.qual4.qual5.qual6 For example:

NorthAmerica.CA.*

Parameters(expressions not allowed):

location -

```
Location
```

Returns: bool_at

MATCHES(text_thostname)

Returns Boolean TRUE if hostname IP address matches the current IP

address.

The hostname string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

hostname -

```
HostName
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

SUBNET(num_atn)

Returns IP Address Value after applying the subnet which can

take values between 0 and 32. For Example: CLIENT.IP.SRC.SUBNET(24) will give 192.168.1.0 for 192.168.1.

Parameters(expressions not allowed):

n -

```
Subnet Value
```

Returns: ip_address_at

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

TYPECAST_IPV6_ADDRESS_AT

Typecast IPv4 address to its "IPv4-mapped IPv6 address"

representation.

Returns: ipv6_address_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert ip_address_at to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

TYPECAST_TEXT_T

Typecast the IP Address Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

TYPECAST_UNSIGNED_LONG_AT

Typecast the IPv4 address value to an Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

IP_ADDRESS_PAIR_T

Provides operations for IP address pair. Takes pair of port (src,dest) as

input.

This topic lists the expressions that are provided by this class.

EQ(ip_address_ati)

Returns Boolean TRUE if either source IP or Destination IP is same as the

argument. Example : CONNECTION.IP.EQ(127.0.0.1)

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

NE(ip_address_ati)

Returns Boolean TRUE if neither source IP nor Destination IP is same as the

argument. Example : CONNECTION.IP.NE(127.0.0.1)

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: bool_at

IP_ADDRESS_T

Represents an IP address present in the 10.100.200.1

format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, ip_address_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

IP address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPLY_MASK**(ip_address_ati)

Returns IP Address Value after applying subnet mask which

can take values between 0.0.0.0 and 255.255.255.255.

Parameters(expressions not allowed):

i -

```
IP Address
```

Returns: ip_address_at

*** APPLY_MASK**(ip_address_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

*** APPLY_MASK**(num_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

*** APPLY_MASK**(time_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

*** APPLY_MASK**(unsigned_long_atmask)

Returns IP Address Value after applying subnet mask

which can take values between 0 and 0xffffffffUL.

Parameters(expressions not allowed):

mask -

```
Subnet Mask
```

Returns: ip_address_at

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the IP address value is same as the

argument. For example: CLIENT.IP.DST.EQ(10.100.10.100)

Parameters:

i -

```
IP Address
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GET1**

Returns a Number Value 10 if the IP address value is 10.100.200.1

Returns: num_at

*** GET2**

Returns a Number Value 100 if the IP address value is 10.100.200.1

Returns: num_at

*** GET3**

Returns a Number Value 200 if the IP address value is 10.100.200.1

Returns: num_at

*** GET4**

Returns a Number Value 1 if the IP address value is 10.100.200.1

Returns: num_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"
```

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

```
Comma: ","


Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IN_SUBNET**(ip_subnet s)

Returns Boolean TRUE if the IP address value is in the

subnet specified by the argument.

For example: CLIENT.IP.DST.IN_SUBNET(10.100.10.100/18)

Parameters(expressions not allowed):

s -

```
IP Subnet</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>IPREP_IS_MALICIOUS</b></
p><div><p>IP Reputation.
```

Returns TRUE if the client IP is categorized as malicious

else returns FALSE.

Returns: bool_at
*** IPREP_THREAT_CATEGORY**(ip_protocol_iprep_category_ecategory)

IP Reputation.

Returns TRUE if the client IP is categorized as malicious

and threat category matches category argument.

else returns FALSE.

Parameters(expressions not allowed):

category -

```
is IP category</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of boolean type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of double data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_IPV6</b></p><div><p>Returns TRUE or
FALSE depending on whether this is an IPv6 client.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LAT_LONG**

Results in text_t containing location as cordinates.

Location will be of the form:

Latitude,Longitude

Expression example: CLIENT.IP.SRC.LAT_LONG

For example: 36,-119

Returns: text_t
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION</b></
p><div><p>Results in list_t containing location.
```

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

Expression example: CLIENT.IP.SRC.LOCATION

For example: NorthAmerica.US....

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LOCATION**(num_atn)

Results in list_t containing location.

Location will be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

Expression example: CLIENT.IP.SRC.LOCATION(2)

For example: NorthAmerica.US

Parameters(expressions not allowed):

n -

```
is the level up to which location will be
```

returned, the first qualifier is numbered 1.
Returns: list_t
*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
```

```
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MATCHES_LOCATION</b>(<a href="text_t.html">text_t</a>location)</
p><div><p>Results in Boolean TRUE
```

value if the location of the IP matches the location argument.

Location string can be of the form:

qual1.qual2.qual3.qual4.qual5.qual6

For example:

NorthAmerica.CA.*

Parameters(expressions not allowed):

location -

```
Location</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MATCHES</b>(<a
href="text_t.html">text_t</a>hostname)</p><div><p>Returns Boolean TRUE if
hostname IP address matches the current IP
```

address.

The hostname string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

hostname -

```
HostName</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the maximum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="ip_address_t.html">ip_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http%3a//" will be decoded to "http://"
2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will

both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"

3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="ip_address_t.html">ip_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED16_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
16-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBNET**(num_atn)

Returns IP Address Value after applying the subnet which can

take values between 0 and 32.

For Example: CLIENT.IP.SRC.SUBNET(24) will give 192.168.1.0 for 192.168.1.

Parameters(expressions not allowed):

n -

```
Subnet Value</p></div><p><b>Returns: </b><a
href="ip_address_at.html">ip_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUBSTR_ANY</b>(<a href="text_t.html">text_t</
a>patset_name)</p><div><p>Select any one sub-string that matches any
string
```

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Typecast IPv4 address to its "IPv4-mapped IPv6 address"

representation.

Returns: ipv6_address_at
*** TYPECAST_IPV6_ADDRESS_T**

Convert text_t to ipv6_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_t : This object represents an IPv6 address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
TYPECAST_IP_ADDRESS_AT

Convert ip_address_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t

TYPECAST_NUM_AT

Convert ip_address_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
ip_address_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Typecast the Number Value to a Time
value.</p><p><b>Returns: </b><a href="time_at.html">time_at</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_T</b></
p><div><p>Convert text_t to time_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_t : This object treats a Text object as a date string. The formats
```

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the IPv4 address value to an Unsigned Long value.

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
```

```
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the

Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t
*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)


returns the &lt;title&gt; element in the document's


&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)
```

```
  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

          person:{

            name:'Milton'

          }

      }
```

```
      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the

    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                    XPath Boolean Expression</p></div><p><b>Returns:
```

```
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


     This will convert the string '36' string to double if it is


     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)



      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)
```

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book&gt;&lt;creator&gt;&lt;person&gt;&lt;name&gt;Milton&lt;/name&gt;&lt;/
person&gt;&lt;/creator&gt;&lt;title&gt;Paradise Lost&lt;/title&gt;&lt;/
book&gt;HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book&gt;&lt;creator&gt;&lt;person&gt;&lt;name&gt;Milton&lt;/name&gt;&lt;/
person&gt;&lt;/creator&gt;&lt;title&gt;Paradise Lost&lt;/title&gt;&lt;price&gt;36&lt;/
price&gt;&lt;/book&gt;HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

IP_PROTOCOL_IPREP_CATEGORY_E

This object represents threat category.

This topic lists the expressions that are provided by this class.

Enumerations

BOTNETS

CLOUD_PROVIDERS

DOS

MOBILE_THREATS

NETWORK

PHISHING

PROXY

REPUTATION

SCANNERS

SPAM_SOURCES

WEB_ATTACKS

WINDOWS_EXPLOITS

IP_PROTOCOL_TYPE_E

This object represents one of the IP Protocols.

This topic lists the expressions that are provided by this class.

Enumerations

AH

ESP

FRAGMENT

GRE

ICMP

ICMPV6

IPIP

OSPFIGP

OSPFPREF

PROTOCOL_UNKNOWN

TCP

UDP

VRRP

Expressions

EQ(ip_protocol_type_eprotocol)

The result is Boolean TRUE if the input argument is equal to

the Protocol represented by this object.

Parameters(expressions not allowed):

protocol -

```
Protocol Type
```

Returns: bool_at

IP_PROTOCOL_TYPE_T

This object provides Text operations and Protocol Enum

operations on the Request Protocol.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from ip_protocol_type_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(ip_protocol_type_eprotocol)

The result is Boolean TRUE if the input argument is equal to

the Protocol represented by this object.

Parameters(expressions not allowed):

protocol -

```
Protocol Type
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert ip_protocol_type_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

IP_REQ_PROTOCOL_T

IP Protocol Request.

This topic lists the expressions that are provided by this class.

DST

Returns the destination IP of the current packet.

Returns: ip_address_at

PROTOCOL

Returns the Protocol of the current packet.

Returns: ip_protocol_type_t

SRC

Returns the source IP of the current packet.

Returns: ip_address_at

IP_RES_PROTOCOL_T

IP Protocol Response.

This topic lists the expressions that are provided by this class.

DST

Returns the destination IP of the current packet.

Returns: ip_address_at

PROTOCOL

Returns the Protocol of the current packet.

Returns: ip_protocol_type_t

SRC

Returns the source IP of the current packet.

Returns: ip_address_at

KEY_USAGE_E

Defines the bits that are allowed in the Key Usage extension of the X509 certificate.

This topic lists the expressions that are provided by this class.

Enumerations

CRLSIGN

DATAENCIPHERMENT

DECIPHERONLY

DIGITAL_SIGNATURE

ENCIPHERONLY

KEYAGREEMENT

KEYCERTSIGN

KEYENCIPHERMENT

NONREPUDIATION

LIST_AT

List Interface

This topic lists the expressions that are provided by this class.

LIST_T

Provides operations where it assumes that the underlying

text is a single character delimited list.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, list_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_ELEMENT**(text_ts)

Returns Boolean TRUE if the given string argument is a

component of the list. The argument string must not contain

the list delimiter character. The IGNORECASE and NOIGNORECASE

textmode settings are honored and can be used to make the result

case insensitive or case sensitive. The URLENCODED and NOURLENCODED

textmode do not affect the result of this operation.

Parameters(expressions not allowed):

s -

```
List Component
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Returns a Number Value giving the count of components

present in the list. The number of components is equal to

the number of delimiters plus one. If the last list

component is of zero length or is all white space then the

number of components is equal to the number of delimiters.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET**(uint i)

Operate as Text on the list component specified by the

index present in the argument. The list component index is

0-based. If the argument index is greater than the number of

components in the list then the selected text will have zero

length.

Parameters(expressions not allowed):

i -

```
Index
```

Returns: text_t

*** GET_REVERSE**(uint i)

Operate as Text on the list component specified by the

reverse index present in the argument. The list component

reverse index is 0-based. The reverse index 0 will select

the last component, reverse index 1 will select the last but

one component and so on. If the argument index is greater

than the number of components in the list then the selected

text will have zero length.

If the last component of the list is of zero length or has

all white spaces then GET_REVERSE(0) starts at the last but

one component.

Parameters(expressions not allowed):

i -

```
Index
```

Returns: text_t

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET**(text_ts)

Operate as Text on the list component given by the string

argument. The argument string must not contain

the list delimiter character. The IGNORECASE and NOIGNORECASE

textmode settings are honored and can be used to make the result

case insensitive or case sensitive. The URLENCODED and NOURLENCODED

textmode do not affect the result of this operation.

Parameters(expressions not allowed):

s -

```
Value
```

Returns: text_t

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

IGNORE_EMPTY_ELEMENTS

This method ignores the empty elements in the list. If the

element delimiter in the list is , then the following list

```
a=10,,b=11, ,c=89
```

has an empty element following a=10. But the element

following b=11 is not considered an empty element.

For example: If HTTP Request packet contains the header

"Cust_Header" as

Cust_Header : 123,,24, ,15

then HTTP.REQ.HEADER("Cust_Header").TYPECAST_LIST_T(',').

IGNORE_EMPTY_ELEMENTS.COUNT gives 4, whereas

HTTP.REQ.HEADER("Cust_Header").TYPECAST_LIST_T(','). .COUNT

gives 5.

Returns: list_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a href="list_t.html">list_t</
a></p></div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377. For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal. For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal. For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>SORT</b></p><div><p>Sorts
the elements of the list.
```

For example, if the input list is 'b,a,c,d,f,e', then the result

will be 'a,b,c,d,e,f'.

Returns: list_t
*** STARTSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text starts with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert list_t to
text_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (text_t : Designates any piece
of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```


String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
  identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing.</p><p>Parameters(expressions not allowed):</p><div><p>xpathex - XPath Expression</p></div><p>Returns: text_t</p></div></div></div><div><div><p>* XPATH(xpath_bool xpathex)</p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

MAC_ADDRESS_AT

Provides operations for the 48-bit MAC address value.

This topic lists the expressions that are provided by this class.

APPEND(text_tstr)

Appends the given string to the string representation of

the current MAC address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

EQ(mac_address_atm)

Compares the MAC address of the packet source ethernet interface against the MAC address

designated in mac, and returns TRUE if they match.

Parameters(expressions not allowed):

m -

```
MAC Address
```

Returns: bool_at

GET1

Returns the hexadecimal code of the first segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 12.

So, if the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET1.EQ(0xd6) will return true.

Returns: num_at

GET2

Returns the hexadecimal code of the second segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 34.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET2.EQ(0x96) will return true.

Returns: num_at

GET3

Returns the hexadecimal code of the third segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 56.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET3.EQ(0xa1) will return true.

Returns: num_at

GET4

Returns the hexadecimal code of the fourth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 78.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET4.EQ(0xa8) will return true.

Returns: num_at

GET5

Returns the hexadecimal code of the fifth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 9a.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET5.EQ(0x89) will return true.

Returns: num_at

GET6

Returns the hexadecimal code of the sixth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns bc.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET6.EQ(0xe4) will return true.

Returns: num_at

TYPECAST_TEXT_T

Typecast the MAC Address Value to its string representation.

Returns: text_t

MAC_ADDRESS_T

Represents a MAC address present in the 12:34:56:78:9a:bc

format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, mac_address_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of

the current MAC address.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(mac_address_atm)

Compares the MAC address of the packet source ethernet interface against the MAC address

designated in mac, and returns TRUE if they match.

Parameters(expressions not allowed):

m -

```
MAC Address
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET1**

Returns the hexadecimal code of the first segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 12.

So, if the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET1.EQ(0xd6) will return true.

Returns: num_at

*** GET2**

Returns the hexadecimal code of the second segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 34.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET2.EQ(0x96) will return true.

Returns: num_at

*** GET3**

Returns the hexadecimal code of the third segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 56.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET3.EQ(0xa1) will return true.

Returns: num_at

*** GET4**

Returns the hexadecimal code of the fourth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 78.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET4.EQ(0xa8) will return true.

Returns: num_at

*** GET5**

Returns the hexadecimal code of the fifth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns 9a.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET5.EQ(0x89) will return true.

Returns: num_at

*** GET6**

Returns the hexadecimal code of the sixth segment of the destination MAC address. For example,

if the MAC address is 12:34:56:78:9a:bc, returns bc.

So, If the Client's MAC address is d6:96:a1:a8:89:e4 then

CLIENT.ETHER.SRCMAC.GET6.EQ(0xe4) will return true.

Returns: num_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="mac_address_t.html">mac_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377. For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal. For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal. For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="mac_address_t.html">mac_address_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6
```

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

```
<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at

*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert mac_address_t to mac_address_at. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -
```

```
                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -
```

quote-character. Returns: nvlist_t * TYPECAST_RADIUS_AVP_T Convert text_t to radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_t : Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

destination type. (tcp_options_t : TCP Protocol Options) Returns:
tcp_options_t </
div> * TYPECAST_TCP_OPTIONS_TYPE_T</
p> Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p> Returns: <a
href="tcp_options_type_t.html">tcp_options_type_t </
div> * TYPECAST_TCP_REPEATER_OPTION_T</
p> Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater) Returns: <a
href="tcp_repeater_option_t.html">tcp_repeater_option_t </
div> TYPECAST_TEXT_T Convert
mac_address_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.) Returns: <a
href="text_t.html">text_t * </
b>TYPECAST_TIME_AT Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
```

```
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from

beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;

- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by
```

```
      /Book/creator"

      in the first 1000 bytes of the body which is

        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost'
```

```
    }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:

  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)
```

```
      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.

    Since this is a node-set the corresponding string

    value will be returned which is "MiltonParadise
```

```
      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is
```

```
      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
```

```
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

MAP_VAR_T

Represents a MAP variable.

This topic lists the expressions that are provided by this class.

VALUECOUNT

Returns an unsigned long count of the current number of

values in the map variable instance.

Returns: unsigned_long_at

VALUEEXISTS(text_tvalue_selector)

Returns true if there is a value in the variable instance

selected by the value_selector expression; i.e., a key for a

map. Otherwise returns false.

Parameters:

value_selector -

```
Key
```

Returns: bool_at

MEDIA_T

Provides expressions to anaylze media content

This topic lists the expressions that are provided by this class.

IS_AUDIO

Returns TRUE if message is audio

Returns: bool_at

IS_DASH_STREAM

Returns TRUE if message is DASH audio/video stream.

Returns: bool_at

IS_HLS_STREAM

Returns TRUE if message is HLS audio/video stream.

Returns: bool_at

IS_VIDEO

Returns TRUE if message is video

Returns: bool_at

MONITOR_T

Operates on a monitor

This topic lists the expressions that are provided by this class.

STATUS

Return the status of the monitor in string.

Returns UP/DOWN as per the monitor status.

Returns: text_t

MSSQL_CLIENT_T

Operates on MSSQL Client Connection

This topic lists the expressions that are provided by this class.

DATABASE

Returns the Database name associated with the client.

Returns: text_t

OPTIONFLAGS1

Returns optionflags1 from Client Capabilities

Returns: num_at

OPTIONFLAGS2

Returns optionflags2 from Client Capabilities

Returns: num_at

OPTIONFLAGS3

Returns optionflags3 from Client Capabilities

Returns: num_at

TYPEFLAGS

Returns typeflags from Client Capabilities

Returns: num_at

USER

Returns the Username associated with the client.

Returns: text_t

MSSQL_PROTOCOL_T

Models the MSSQL message types.

This topic lists the expressions that are provided by this class.

CLIENT

Operates on MSSQL Client.

Returns: mssql_client_t

REQ

Operates on MSSQL Request.

Returns: mssql_req_t

RES

Operates on MSSQL Response.

Returns: mssql_res_t

MSSQL_REQ_COMMAND_E

This object represents one of the SQL Request Commands.

This topic lists the expressions that are provided by this class.

Enumerations

ATTENTION

BULKDATA

COM_UNKNOWN

LOGIN

PRELOGIN

PRETDS7

QUERY

RESPONSE

RPC

SSPI

TRANSMGR

Expressions

EQ(mssql_req_command_em)

The result is Boolean TRUE if the input argument is equal to

the MSSQL Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Request Command Name
```

Returns: bool_at

NE(mssql_req_command_em)

The result is Boolean TRUE if the input argument is not equal to

the MSSQL Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Request Command Name
```

Returns: bool_at

MSSQL_REQ_QUERY_T

Operates on MSSQL Request Query.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sql_query_t.

COMMAND

String for the first keyword of the query.

Returns: text_t

SIZE

Integer representing the Size of the request query. Similar to

CONTENT_LENGTH property of HTTP.

Returns: num_at

TEXT

Returns string covering the entire query.

Returns: text_t

TEXT(uint n)

Returns a string covering the first n bytes of query. Like

HTTP.BODY(n)

Parameters:

n -

```
Query Length
```

Returns: text_t

TYPECAST_SQL_QUERY_T

Convert mssql_req_query_t to sql_query_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sql_query_t : Describes a SQL query.)

Returns: sql_query_t

MSSQL_REQ_RPC_T

Operates on MSSQL RPC request.

This topic lists the expressions that are provided by this class.

BODY

Returns the body of the SQL request as a String in the form of

parameters represented as "a=b" clauses separated by commas where

"a" is the RPC parameter name and "b" is its value.

Returns: text_t

BODY(uint n)

Returns the body of the SQL request as a String in the form of

parameters represented as "a=b" clauses separated by commas where

"a" is the RPC parameter name and "b" is its value. The parameters

lookup is done in first n-bytes of the request, skipping the sql-header.

It returns only complete pairs.

Parameters:

n -

```
(RPC body len)
```

Returns: text_t

IS_PROCID

Does the RPC request contain ProcID? If it returns false

it indicates the RPC contains a RPC Name.

Returns: bool_at

NAME

String for the name of the procedure.

Returns: text_t

PROCID

Integer representing the ProcID of the RPC request.

Returns: num_at

MSSQL_REQ_T

Operates on MSSQL Request.

This topic lists the expressions that are provided by this class.

COMMAND

Enum for the type of request command. The different values of the

ENUM will be QUERY, RESPONSE, RPC, ATTENTION etc.

Returns: mssql_req_command_e

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

QUERY

returns a Request Query Object.

Returns: mssql_req_query_t

RPC

returns a Request RPC Object.

Returns: mssql_req_rpc_t

MSSQL_RES_DONE_T

mssql response done object

This topic lists the expressions that are provided by this class.

ROWCOUNT

number of rows affected by INSERT/UPDATE/DELETE queries.

Returns: unsigned_long_at

STATUS

status of the done token of the response.

The Status MUST be a bitwise 'OR' of the following:

0x00: DONE_FINAL (this DONE is the final DONE in the request).

0x1: DONE_MORE (this DONE message is not the final DONE message in

```
the result batch. Subsequent statements to follow).
```

0x2: DONE_ERROR (on error occurred on the current command. A

```
preceding ERROR token SHOULD be sent when this bit is set).
```

0x4: DONE_INXACT (a transaction is in progress).

0x10: DONE_COUNT (the DoneRowCount value is valid. This is used to

```
 distinguish between a valid value of 0 for DoneRowCount or just



 an initialized variable).
```

0x20: DONE_ATTN (the DONE message is a server acknowledgement of a

```
client ATTENTION message).
```

0x100: DONE_SRVERROR (used in place of DONE_ERROR when an error

```
 occurred on the current command, which is severe enough to



 require the result set,    if any, to be discarded).
```

Returns: num_at

MSSQL_RES_ERROR_T

mssql response error object

This topic lists the expressions that are provided by this class.

MESSAGE

String for the error message field of the error response.

Returns: text_t

NUM

Error number field of the error response.

Returns: num_at

STATE

Integer for the error state field of the error response, used as a

modifier to the error number.

Returns: num_at

MSSQL_RES_FIELD_DATATYPE_E

This object represents one of the MSSQL response fields datatype.

This topic lists the expressions that are provided by this class.

Enumerations

BIGCHRTYPE

BIGVARCHRTYPE

CHRTYPE

DATENTYPE

INTNTYPE

NCHRTYPE

NVARCHRTYPE

TIMENTYPE

UNKNOWN

VARCHRTYPE

Expressions

EQ(mssql_res_field_datatype_em)

The result is Boolean TRUE if the input argument is equal to

the MSSQL fields datatype value represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Response Field's datatype
```

Returns: bool_at

NE(mssql_res_field_datatype_em)

The result is Boolean TRUE if the input argument is not equal to

the MSSQL field's datatype value represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Response Field's datatype
```

Returns: bool_at

MSSQL_RES_FIELD_T

Returns mssql response' field object

This topic lists the expressions that are provided by this class.

DATATYPE

returns datatype of particular column (table attribute).

Returns: mssql_res_field_datatype_e

LENGTH

returns length of particular column (table attribute).

Returns: num_at

NAME

returns name of particular column (table attribute).

Returns: text_t

MSSQL_RES_ROW_T

Returns MSSQL Response Row object

This topic lists the expressions that are provided by this class.

DOUBLE_ELEM(uint j)

returns double value from j'th column of i'th row of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: double_at

IS_NULL_ELEM(uint j)

check if j'th column of i'th row of the table is NULL.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: bool_at

NUM_ELEM(uint j)

Returns integer value from j'th column of i'th row of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: num_at

TEXT_ELEM(uint j)

Returns string value from j'th column of i'th rows of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: text_t

MSSQL_RES_T

Operates on MSSQL Response.

This topic lists the expressions that are provided by this class.

ATLEAST_ROWS_COUNT(uint i)

Checks if response has at least this many rows.

Parameters(expressions not allowed):

i -

```
number of rows
```

Returns: bool_at

DONE

This object provides for the ability to work on Done Token pkt of

Response.

Returns: mssql_res_done_t

ERROR

returns Error object representing error number, error msg, etc.

Returns: mssql_res_error_t

FIELD(uint i)

This object provides for the ability to work on individual Field

pkts(columns) of the Response.

Parameters(expressions not allowed):

i -

```
column number
```

Returns: mssql_res_field_t

ROW(uint i)

This object provides for the ability to work on individual Row pkts

of the Response.

Parameters(expressions not allowed):

i -

```
row number
```

Returns: mssql_res_row_t

TYPE

Enum for the response type.

Its values can be ERROR, OK, and RESULT_SET

Returns: mssql_res_type_e

MSSQL_RES_TYPE_E

This object represents MSSQL Reponse Types.

This topic lists the expressions that are provided by this class.

Enumerations

ERROR

OK

RESULT_SET

Expressions

EQ(mssql_res_type_em)

The result is Boolean TRUE if the input argument is equal to

the MSSQL Reponse Type represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Response type Name
```

Returns: bool_at

NE(mssql_res_type_em)

The result is Boolean TRUE if the input argument is not equal to

the MSSQL Response Type represented by this object.

Parameters(expressions not allowed):

m -

```
MSSQL Response type Name
```

Returns: bool_at

MYSQL_CHARSET_E

This object represents one of the MYSQL Charset.

This topic lists the expressions that are provided by this class.

Enumerations

ARMSCII8_BIN

ARMSCII8_GENERAL_CI

ASCII_BIN

ASCII_GENERAL_CI

CP1250_BIN

CP1250_CROATIAN_CI

CP1250_CZECH_CS

CP1250_GENERAL_CI

CP1251_BIN

CP1251_BULGARIAN_CI

CP1251_GENERAL_CI

CP1251_GENERAL_CS

CP1251_UKRAINIAN_CI

CP1256_BIN

CP1256_GENERAL_CI

CP1257_BIN

CP1257_GENERAL_CI

CP1257_LITHUANIAN_CI

CP850_BIN

CP850_GENERAL_CI

CP852_BIN

CP852_GENERAL_CI

CP866_BIN

CP866_GENERAL_CI

DEC8_BIN

DEC8_SWEDISH_CI

GEOSTD8_BIN

GEOSTD8_GENERAL_CI

GREEK_BIN

GREEK_GENERAL_CI

HEBREW_BIN

HEBREW_GENERAL_CI

HP8_BIN

HP8_ENGLISH_CI

INVAL_CHARSET

KEYBCS2_BIN

KEYBCS2_GENERAL_CI

KOI8R_BIN

KOI8R_GENERAL_CI

KOI8U_BIN

KOI8U_GENERAL_CI

LATIN1_BIN

LATIN1_DANISH_CI

LATIN1_GENERAL_CI

LATIN1_GENERAL_CS

LATIN1_GERMAN1_CI

LATIN1_GERMAN2_CI

LATIN1_SPANISH_CI

LATIN1_SWEDISH_CI

LATIN2_BIN

LATIN2_CROATIAN_CI

LATIN2_CZECH_CS

LATIN2_GENERAL_CI

LATIN2_HUNGARIAN_CI

LATIN5_BIN

LATIN5_TURKISH_CI

LATIN7_BIN

LATIN7_ESTONIAN_CS

LATIN7_GENERAL_CI

LATIN7_GENERAL_CS

MACCE_BIN

MACCE_GENERAL_CI

MACROMAN_BIN

MACROMAN_GENERAL_CI

SWE7_BIN

SWE7_SWEDISH_CI

UTF8_BIN

UTF8_CZECH_CI

UTF8_DANISH_CI

UTF8_ESPERANTO_CI

UTF8_ESTONIAN_CI

UTF8_GENERAL_CI

UTF8_HUNGARIAN_CI

UTF8_ICELANDIC_CI

UTF8_LATVIAN_CI

UTF8_LITHUANIAN_CI

UTF8_PERSIAN_CI

UTF8_POLISH_CI

UTF8_ROMANIAN_CI

UTF8_ROMAN_CI

UTF8_SLOVAK_CI

UTF8_SLOVENIAN_CI

UTF8_SPANISH2_CI

UTF8_SPANISH_CI

UTF8_SWEDISH_CI

UTF8_TURKISH_CI

UTF8_UNICODE_CI

Expressions

EQ(mysql_charset_em)

The result is Boolean TRUE if the input argument is equal to

the MYSQL charset value represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Character Set Name
```

Returns: bool_at

NE(mysql_charset_em)

The result is Boolean TRUE if the input argument is not equal to

the MYSQL charset value represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Character Set Name
```

Returns: bool_at

MYSQL_CLIENT_T

This Object operates on client part of the MYSQL Connection.

This topic lists the expressions that are provided by this class.

CAPABILITIES

Returns the Client Flags. The list of possible flag values is in the description of the Handshake Initialisation Packet, for server_capabilities. For some of the bits, the server passed "what it's capable of". The client leaves some of the bits on, adds others and passes back to the server. One important flag is: whether compression is desired. Another interesting one is CLIENT_CONNECT_WITH_DB, which shows the presence of the optional databasename).

Returns: num_at

CHAR_SET

Returns the character set of the client.

Returns: mysql_charset_e

DATABASE

Returns the Database name of the client.

Returns: text_t

USER

Returns the User associated with the client.

Returns: text_t

MYSQL_PROTOCOL_T

Models the MYSQL message types.

This topic lists the expressions that are provided by this class.

CLIENT

Operates on MYSQL Client.

Returns: mysql_client_t

REQ

Operates on MYSQL Request.

Returns: mysql_req_t

RES

Operates on MYSQL Response.

Returns: mysql_res_t

MYSQL_REQ_COMMAND_E

This object represents one of the SQL Request Commands.

This topic lists the expressions that are provided by this class.

Enumerations

BINLOG_DUMP

CHANGE_USER

CONNECT

CONNECT_OUT

CREATE_DB

DEBUG

DELAYED_INSERT

DROP_DB

FIELD_LIST

INIT_DB

PING

PROCESS_INFO

PROCESS_KILL

QUERY

QUIT

REFRESH

REGISTER_SLAVE

SET_OPTION

SHUTDOWN

SLEEP

STATISTICS

STMT_CLOSE

STMT_EXECUTE

STMT_FETCH

STMT_PREPARE

STMT_RESET

STMT_SEND_LONG_DATA

TABLE_DUMP

TIME

Expressions

EQ(mysql_req_command_em)

The result is Boolean TRUE if the input argument is equal to

the MYSQL Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Request Command Name
```

Returns: bool_at

NE(mysql_req_command_em)

The result is Boolean TRUE if the input argument is not equal to

the MYSQL Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Request Command Name
```

Returns: bool_at

MYSQL_REQ_QUERY_T

Describes a MySQL query.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sql_query_t.

COMMAND

String for the first keyword of the query.

Returns: text_t

SIZE

Integer representing the Size of the request query. Similar to

CONTENT_LENGTH property of HTTP.

Returns: num_at

TEXT

Returns a string covering the entire query.

Returns: text_t

TEXT(uint n)

Returns a string covering the first n bytes of query. Like

HTTP.BODY(n).

Parameters:

n -

```
first bytes in query
```

Returns: text_t

TYPECAST_SQL_QUERY_T

Convert mysql_req_query_t to sql_query_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sql_query_t : Describes a SQL query.)

Returns: sql_query_t

MYSQL_REQ_T

Operates on MYSQL Request.

This topic lists the expressions that are provided by this class.

COMMAND

Enum for the type of request command. The different values of the

ENUM will be QUERY, INIT_DB, etc.

Returns: mysql_req_command_e

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

QUERY

returns Request Query Object.

Returns: mysql_req_query_t

MYSQL_RES_ERROR_T

mysql response error.

This topic lists the expressions that are provided by this class.

MESSAGE

String for the message field of the error response.

Returns: text_t

NUM

Error number field of the error response.

Returns: num_at

SQLSTATE

String for the SqlState field of the error response.

The server translates error number values to sqlstate values.

Returns: text_t

MYSQL_RES_FIELD_DATATYPE_E

This object represents one of the MYSQL response field datatype.

This topic lists the expressions that are provided by this class.

Enumerations

BIT

BLOB

DATE

DATETIME

DECIMAL

DOUBLE

ENUM

FLOAT

GEOMETRY

INT24

LONG

LONGLONG

LONG_BLOB

MEDIUM_BLOB

NEWDATE

NEWDECIMAL

NULL

SET

SHORT

STRING

TIME

TIMESTAMP

TINY

TINY_BLOB

UNKNOWN

VARCHAR

VAR_STRING

YEAR

Expressions

EQ(mysql_res_field_datatype_em)

The result is Boolean TRUE if the input argument is equal to

the MYSQL fields datatype value represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Response Field's datatype
```

Returns: bool_at

NE(mysql_res_field_datatype_em)

The result is Boolean TRUE if the input argument is not equal to

the MYSQL field's datatype value represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Response Field's datatype
```

Returns: bool_at

MYSQL_RES_FIELD_T

Returns mysql response field object. It allows to access properties of

individual attributes of the query response.

This topic lists the expressions that are provided by this class.

CATALOG

returns the Catalog name of the column.

Returns: text_t

CHAR_SET

returns charset of the column.

Returns: mysql_charset_e

DATATYPE

returns datatype of the column.

possible values in Hex

0x00 DECIMAL

0x01 TINY

0x02 SHORT

0x03 LONG

0x04 FLOAT

0x05 DOUBLE

0x06 NULL

0x07 TIMESTAMP

0x08 LONGLONG

0x09 INT24

0x0a DATE

0x0b TIME

0x0c DATETIME

0x0d YEAR

0x0e NEWDATE

0x0f VARCHAR (new in MySQL 5.0)

0x10 BIT (new in MySQL 5.0)

0xf6 NEWDECIMAL (new in MYSQL 5.0)

0xf7 ENUM

0xf8 SET

0xf9 TINY_BLOB

0xfa MEDIUM_BLOB

0xfb LONG_BLOB

0xfc BLOB

0xfd VAR_STRING

0xfe STRING

0xff GEOMETRY

Returns: mysql_res_field_datatype_e

DB

returns the database name of the column.

Returns: text_t

DECIMALS

The number of positions after the decimal point if the type is

DECIMAL or NUMERIC.

Returns: num_at

FLAGS

returns flag values of the column.

The possible flag values (in hexadec)

0001 NOT_NULL_FLAG

0002 PRI_KEY_FLAG

0004 UNIQUE_KEY_FLAG

0008 MULTIPLE_KEY_FLAG

0010 BLOB_FLAG

0020 UNSIGNED_FLAG

0040 ZEROFILL_FLAG

0080 BINARY_FLAG

0100 ENUM_FLAG

0200 AUTO_INCREMENT_FLAG

0400 TIMESTAMP_FLAG

0800 SET_FLAG

Returns: num_at

LENGTH

returns length of the column. The value given here may be

larger than the actual length, for example an instance of a

VARCHAR(2) column may have only 1 character in it.

Returns: num_at

NAME

returns alias name of the column (table attribute).

Returns: text_t

ORIGINAL_NAME

returns alias name of the column (table attribute).

Returns: text_t

ORIGINAL_TABLE

returns table name of the column.

Returns: text_t

TABLE

returns alias table name of the column.

Returns: text_t

MYSQL_RES_OK_T

mysql response ok object. Allows to access the ok packet properties of the

response.

This topic lists the expressions that are provided by this class.

AFFECTED_ROWS

number of rows affected by INSERT/UPDATE/DELETE.

Returns: unsigned_long_at

INSERT_ID

If the statement (query) generated any AUTO_INCREMENT number, it is

returned here. Otherwise this field contains 0.

Returns: unsigned_long_at

MESSAGE

returns text message after INSERT etc queries.

Returns: text_t

STATUS

Client can use the server status to check if command was inside a

transaction. Server status contains a bit string, each bit

corresponds to these fields:

IN TRANSACTION, AUTO_COMMIT, MORE RESULTS, MULTI QUERY, BAD INDEX

USED, NO INDEX USED, CURSOR EXISTS, LAST ROW SEEN, DATABASE DROPPED,

and NO BACKSLASH ESCAPES.

Returns: num_at

WARNING_COUNT

returns the warning_count attribute of OK Packet.

Returns: num_at

MYSQL_RES_ROW_T

mysql response row object.

This topic lists the expressions that are provided by this class.

DOUBLE_ELEM(uint j)

returns double value from j'th column of i'th rows of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column

Parameters(expressions not allowed):

j -

```
column number
```

Returns: double_at

IS_NULL_ELEM(uint j)

check if j'th column of i'th rows of the table is NULL.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column

Parameters(expressions not allowed):

j -

```
column number
```

Returns: bool_at

NUM_ELEM(uint j)

returns int value from j'th column of some i'th rows of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column

Parameters(expressions not allowed):

j -

```
column number
```

Returns: unsigned_long_at

TEXT_ELEM(uint j)

returns string value from j'th column of i'th rows of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column

Parameters(expressions not allowed):

j -

```
column number
```

Returns: text_t

MYSQL_RES_T

Operates on MYSQL Response.

This topic lists the expressions that are provided by this class.

ATLEAST_ROWS_COUNT(uint i)

Checks if response has at least this many rows.

Parameters(expressions not allowed):

i -

```
number of rows
```

Returns: bool_at

ERROR

returns error object representing error number, error msg, etc.

Returns: mysql_res_error_t

FIELDS_COUNT

Returns the number of columns in the response.

Returns: unsigned_long_at

FIELD(uint i)

This object provides for the ability to work on individual Field

pkts of the Response.

Parameters(expressions not allowed):

i -

```
pkt number
```

Returns: mysql_res_field_t

OK

This object provides for the ability to work on OK pkt of response.

Returns: mysql_res_ok_t

ROW(uint i)

This object provides for the ability to work on individual Row pkts

of the Response.

Parameters(expressions not allowed):

i -

```
row number
```

Returns: mysql_res_row_t

TYPE

Enum for the response type.

Its values can be ERROR, OK, and RESULT_SET.

Returns: mysql_res_type_e

MYSQL_RES_TYPE_E

This object represents MYSQL Reponse Types.

This topic lists the expressions that are provided by this class.

Enumerations

ERROR

OK

RESULT_SET

Expressions

EQ(mysql_res_type_em)

The result is Boolean TRUE if the input argument is equal to

the MYSQL Reponse Type represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Response type Name
```

Returns: bool_at

NE(mysql_res_type_em)

The result is Boolean TRUE if the input argument is not equal to

the MYSQL Response Type represented by this object.

Parameters(expressions not allowed):

m -

```
MYSQL Response type Name
```

Returns: bool_at

NUM_AT

Number Value

This topic lists the expressions that are provided by this class.

ABS

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

ADD(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITAND(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITNEG

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITOR(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITXOR(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DIV(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

EQ(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GE(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GT(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LE(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LSHIFT(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LT(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MAX(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MIN(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MOD(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MUL(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

NEG

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

NE(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

RSHIFT(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

SIGNED16_STRING(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

SIGNED32_STRING(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

SIGNED8_STRING

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

SUB(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

TYPECAST_DOUBLE_AT

Typecast the Number Value to a Double value.

Returns: double_at

TYPECAST_IP_ADDRESS_AT

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

*** TYPECAST_NUM_AT**

Typecast the Unsigned Long Value to a Number value.

Returns: num_at

TYPECAST_TEXT_T

Typecast the Number Value to its string representation.

Returns: text_t

TYPECAST_TIME_AT

Typecast the Number Value to a Time value.

Returns: time_at

TYPECAST_UNSIGNED_LONG_AT

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

UNSIGNED16_STRING(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

UNSIGNED8_STRING

Create an unsigned 8-bit binary string from the number.

Returns: text_t

WEEKDAY_STRING

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

WEEKDAY_STRING_SHORT

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

NUM_FORMAT_E

Number Format Enum

This topic lists the expressions that are provided by this class.

Enumerations

DECIMAL

Decimal representation of a positive or a negative integer. White spaces before and after the number are OK

DECIMAL_PREFIX

Decimal representation of a positive of a negative integer followed by arbitrary characters Examples " -123" " 234 " "- 123abc" "234 abc"

HEX

Hexadecimal representation of a positive or a negative integer. White spaces before and after the number are OK

HEX_PREFIX

Hexadecimal representation of a positive integer followed by arbitrary characters Examples "f" "fxyz" " f ff" all represent the number 15.

NUM_T

Provides operations to treat the underlying text as a Number either

in the DECIMAL or the HEX format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
```

```
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +
```

```
"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></
p></div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>SIGNED16_STRING</b>(<a href="endian_e.html">endian_e</a>endianness)</
p><div><p>Create a signed 16-bit binary string from the number.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
```

```
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
```

```
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
```

```
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

TYPECAST_NUM_AT

Convert num_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
```

```
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert num_t to
text_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (text_t : Designates any piece
of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Typecast the Number Value to a Time value.</p><p><b>Returns: </
b><a href="time_at.html">time_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TIME_T</b></p><div><p>Convert text_t
to time_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (time_t : This object
treats a Text object as a date string. The formats
```

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```


supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the

Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```


the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -


```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```


and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t
*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's



&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)
```

```
  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

          person:{

            name:'Milton'

          }

      }
```

```
      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the

    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                    XPath Boolean Expression</p></div><p><b>Returns:
```

```
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)



      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)
```

```
      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

NVLIST_AT

Name Value List Interface

This topic lists the expressions that are provided by this class.

NVLIST_T

Treats a Text object as a name-value list with 2

delimiters. Example of a name-value list is abc=1;def=2.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, nvlist_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Return a Number Value giving the number of name-value

components in the list. If the last Name-Value List

component is of zero length or is all spaces then it is

ignored.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

IGNORE_EMPTY_ELEMENTS

This method ignores the empty elements in the nv-list. If

the element (i.e. the second) delimiter in the nv-list is ;

then the following list

```
a=10;;b=11; ;c=89
```

has an empty element following a=10. But the element

following b=11 is not considered an empty element.

For Example: If HTTP Request packet contains the header

"Cust_Header" as

Cust_Header : a=1;;b=2; ;c=3

then

HTTP.REQ.HEADER("Cust_Header").TYPECAST_NVLIST_T('=',';').

IGNORE_EMPTY_ELEMENTS.COUNT gives 4, whereas

HTTP.REQ.HEADER("Cust_Header").TYPECAST_NVLIST_T('=',';').

.COUNT gives 5.

Returns: nvlist_t

*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at

*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAMES</b></p><div><p>Extracts a
list of only the names from the name-value pair list.
```

For example the list 'a=b,c="d=e,f",g=h' will result in the list

'a,c,g'. Empty elements will be ignored.

Returns: list_t
*** NAME**(uint i)

Operate as Text on the name component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(uint i)</
p><div><p>Returns the element in the list identified by the parameter.
```

The parameter is a 0-based index in to the list. The result

is a string. If the specified index is greater than the

number of components then a zero length string is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the name-
value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NAME_VALUE</b>(<a
```

```
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the name-value component
```

in the list. The name-value component

is specified by the argument by providing the name whose

name-value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>REPLACE_ALL_VALUES</b>(<a
href="text_t.html">text_t</a>value)</p><div><p>Replaces all the values in
each name-value pair with the specified
```

string argument.

For example, given argument string "?" the list 'a=b,c=d,e=f' will

result in the name-value list 'a=?,c=?,e=?'. Quotes will be stripped

from the values being replaced. Empty values will be replaced with

the value provided. Empty elements will be preserved.

Parameters(expressions not allowed):

value -

```
value to replace existing values</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_TEXT_MODE</b>(<a href="textmode_e.html">textmode_e</a>mode)</
p><div><p>The mode setting of a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="nvlist_t.html">nvlist_t</
a></p></div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint
i)</p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
TYPECAST_LIST_T

Treats this name-value pair list as a simple list_t separated

by the same element separator ("delimiter2").

Note that lists do not honor the quote characters, so

any nvlist_t entry that has quotes and a quoted entry includes a

separator will be split at that embedded separator. Empty elements

will not be ignored.

Returns: list_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
```

```
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
```

```
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
```

```
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert nvlist_t
to text_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (text_t : Designates any
piece of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** VALUES**

Extracts a list of only the values from the name-value pair list.

For example the list 'a=b,c=d,e=f' will result in the list

'b,d,f'. Note that lists do not honor the quote characters, so

any nvlist_at entry that has quotes and a quoted entry includes a

separator will be split at that embedded separator. Quote

characters will not be stripped from the values. Empty elements

and empty values will be ignored.

Returns: list_t
*** VALUE**(uint i)

Operate as Text on the value component of the specified

name-value component in the list. The index of the

name-value component given by the argument is 0-based. If

the specified index is greater than the number of components

then a zero length Text object is produced.

Parameters(expressions not allowed):

i -

```
Index</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name)</p><div><p>Operate as Text on the value
component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. The first component that matches the name

is selected. The matching process honors the IGNORECASE and

the NOIGNORECASE text modes. The URLENCODED and the

NOURLENCODED text modes are ignored.

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>VALUE</b>(<a
href="text_t.html">text_t</a>name, uint i)</p><div><p>Operate as Text on
the value component of the specified
```

name-value component in the list. The name-value component

is specified by the argument by providing the name whose

value is needed. In a name-value list the same name can be

repeated multiple times. The Index argument controls which

occurence of the matching name-value component should be

selected. The Index is 0-based. If the Index is too big then

the Text object that is produced will have zero length. The

matching process honors the IGNORECASE and the NOIGNORECASE

text modes. The URLENCODED and the NOURLENCODED text modes

are ignored.

Parameters(expressions not allowed):

name -

```
                        Name</p><p>i -



                        Index</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XML_DECRYPT</b>(<a href="text_t.html">text_t</a>certKeyName)</
p><div><p>Decrypt the selected content, which was encrypted using the W3C
XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by
```

```
      /Book/creator"

      in the first 1000 bytes of the body which is

        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'
```

```
    }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)
```

```
      This will convert the string '36' string to double if it is



      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.

    Since this is a node-set the corresponding string

    value will be returned which is "MiltonParadise
```

```
      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the
```

```
      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)


      This will return true as there is a node called creator


      under the node Book with in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

For the XML file:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t </body></html>

ORACLE_CLIENT_T

This Object operates on client part of the ORACLE Connection.

This topic lists the expressions that are provided by this class.

DATABASE

Returns the Database name of the client.

Returns: text_t

USER

Returns the User associated with the client.

Returns: text_t

ORACLE_PROTOCOL_T

Models the ORACLE message types.

This topic lists the expressions that are provided by this class.

CLIENT

Operates on ORACLE Client.

Returns: oracle_client_t

REQ

Operates on ORACLE Request.

Returns: oracle_req_t

RES

Operates on ORACLE Response.

Returns: oracle_res_t

ORACLE_REQ_COMMAND_E

This object represents one of the SQL Request Commands.

This topic lists the expressions that are provided by this class.

Enumerations

BINLOG_DUMP

CHANGE_USER

CONNECT

CONNECT_OUT

CREATE_DB

DEBUG

DELAYED_INSERT

DROP_DB

FIELD_LIST

INIT_DB

PING

PROCESS_INFO

PROCESS_KILL

QUERY

QUIT

REFRESH

REGISTER_SLAVE

SET_OPTION

SHUTDOWN

SLEEP

STATISTICS

STMT_CLOSE

STMT_EXECUTE

STMT_FETCH

STMT_PREPARE

STMT_RESET

STMT_SEND_LONG_DATA

TABLE_DUMP

TIME

Expressions

EQ(oracle_req_command_em)

The result is Boolean TRUE if the input argument is equal to

the ORACLE Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
ORACLE Request Command Name
```

Returns: bool_at

NE(oracle_req_command_em)

The result is Boolean TRUE if the input argument is not equal to

the ORACLE Request Command represented by this object.

Parameters(expressions not allowed):

m -

```
ORACLE Request Command Name
```

Returns: bool_at

ORACLE_REQ_QUERY_T

Describes a ORACLE query.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sql_query_t.

COMMAND

String for the first keyword of the query.

Returns: text_t

SIZE

Integer representing the Size of the request query. Similar to

CONTENT_LENGTH property of HTTP.

Returns: num_at

TEXT

Returns a string covering the entire query.

Returns: text_t

TEXT(uint n)

Returns a string covering the first n bytes of query. Like

HTTP.BODY(n).

Parameters:

n -

```
first bytes in query
```

Returns: text_t

TYPECAST_SQL_QUERY_T

Convert oracle_req_query_t to sql_query_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sql_query_t : Describes a SQL query.)

Returns: sql_query_t

ORACLE_REQ_T

Operates on ORACLE Request.

This topic lists the expressions that are provided by this class.

COMMAND

Enum for the type of request command. The different values of the

ENUM will be QUERY, INIT_DB, etc.

Returns: oracle_req_command_e

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

QUERY

returns Request Query Object.

Returns: oracle_req_query_t

ORACLE_RES_ROW_T

Returns ORACLE Response Row object

This topic lists the expressions that are provided by this class.

IS_NULL_ELEM(uint j)

check if j'th column of i'th row of the table is NULL.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: bool_at

NUM_ELEM(uint j)

Returns integer value from j'th column of i'th row of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: num_at

TEXT_ELEM(uint j)

Returns string value from j'th column of i'th rows of the table.

Following C conventions, both index i and j start from 0 only,

hence row i and comlumn j is actually (i+1)th row and (j+1)th column.

Parameters(expressions not allowed):

j -

```
column number
```

Returns: text_t

ORACLE_RES_T

Operates on ORACLE Response.

This topic lists the expressions that are provided by this class.

ATLEAST_ROWS_COUNT(uint i)

Checks if response has at least this many rows.

Parameters(expressions not allowed):

i -

```
number of rows
```

Returns: bool_at

ROW(uint i)

This object provides for the ability to work on individual Row pkts

of the Response.

Parameters(expressions not allowed):

i -

```
row number
```

Returns: oracle_res_row_t

PACKET_T

Designates an expresion that refers to a network packet.

This topic lists the expressions that are provided by this class.

CONNID

Returns the connection ID of current network packet

Returns: num_at

CS_VSERVER

Returns CS vserver

Returns: vserver_t

DSTIP

Returns the destination IP of the current IPv4 packet.

Returns: ip_address_at

DSTIPV6

Returns the destination IP of the current IPv6 packet.

Returns: ipv6_address_at

DSTPORT

Returns the ID of the current packet's TCP/UDP destination port as a number.

Returns: num_at

IDLETIME

Returns connection idle time

Returns: num_at

INTF

Extracts the interface ID of the interface through which the current packet entered the Application Switch, and returns that ID.

Returns: interface_t

IP

Equivalent to SRCIP() || DSTIP()

Returns: ip_address_pair_t

IPV6

Equivalent to SRCIPv6() || DSTIPv6()

Returns: ipv6_address_pair_t

LB_VSERVER

Returns LB vserver

Returns: vserver_t

PORT

equivalent to SRCPORT() || DSTPORT()

Returns: port_pair_t

PPEID

Extracts the ppeid or core ID of the current packet

Returns: num_at

SERVICE_TYPE

Returns connection service type

Returns: service_type_e

SRCIP

Returns the source IP of the current IPv4 packet.

Returns: ip_address_at

SRCIPV6

Returns the source IP of the current IPv6 packet.

Returns: ipv6_address_at

SRCPORT

Returns the ID of the current packet's TCP/UDP source port as a number.

Returns: num_at

SVCNAME

Returns the service name

Returns: text_t

TCPSTATE

Returns TCP connection state

Returns: tcp_state_e

TRAFFIC_DOMAIN_ID

Returns Traffic Domain ID of the current packet

Returns: num_at

VLANID

Returns the ID of the VLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

VXLANID

Returns the VNID of the VXLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

PORT_PAIR_T

provides operations for the TCP/UDP src/dst port

This topic lists the expressions that are provided by this class.

BETWEEN(uint i, uint j)

Returns Boolean TRUE if the source port OR destination port is in between

given range. Example : CONNECTION.PORT.BETWEEN(60, 500)

Parameters(expressions not allowed):

i -

```
Port Number
```

j -

```
Port Number
```

Returns: bool_at

EQ(uint i)

Returns Boolean TRUE if either source port OR destination port is same as the

argument. Example : CONNECTION.PORT.EQ(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

GE(uint i)

Returns Boolean TRUE if the source port OR destination port is greater than or equal to

the argument. Example : CONNECTION.PORT.GE(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

GT(uint i)

Returns Boolean TRUE if the source port OR destination port is greater than

the argument. Example : CONNECTION.PORT.GT(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

LE(uint i)

Returns Boolean TRUE if the source port OR destination port is less than or equal to

the argument. Example : CONNECTION.PORT.LE(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

LT(uint i)

Returns Boolean TRUE if the source port OR destination port is less than

the argument. Example : CONNECTION.PORT.LT(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

NE(uint i)

Returns Boolean TRUE if both the source port AND destination port are not same as

the argument. Example : CONNECTION.PORT.NE(80)

Parameters(expressions not allowed):

i -

```
Port Number
```

Returns: bool_at

RADIUS_AVP_T

Operations that can be performed on the returned AVPs.

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select
```

```
"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

AVPLIST_AFTER

Returns the region after this AVP in radius message (or in other words

returns location at end of this AVP and length covering end

of this AVP to end of radius message.

Returns: text_t

AVPLIST_BEFORE

Returns the region till this AVP in radius message (or in other words

returns location at start of AVPs and length equal to offset of

this AVP).

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

COUNT

Returns the number of occurences of an AVP in radius message.

The system does not keep track of more than 16 AVP occurences of an AVP. The result

will be undefined if there are more than 16 instances of that AVP.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns true if the AVP exists in the message.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
INSTANCE(uint instanceno)

Returns instances of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: radius.REQ.AVP(26). However, subsequent instances must be

retrieved using this method.

INSTANCE(0) retrieves the first instance, INSTANCE(1) retrieves the second

instance, and so on.

For example, to get the second instance, the expression would be:
radius.REQ.AVP(282).INSTANCE(1).

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="radius_avp_t.html">radius_avp_t</a></p></div></div></
div><div><div><p><b>* </b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of boolean type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_DOUBLE_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of double data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean TRUE value if the Text
```

```
is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

LENGTH

Returns the AVP length including AVP header.

If the AVP is not present, it returns length 0. If multiple occurences

of the AVP exist, it returns the length of first occurence.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyVAlue> element with

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

> the binary signature.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="radius_avp_t.html">radius_avp_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="radius_avp_t.html">radius_avp_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
radius_avp_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

VALUE

Returns the value of first instance of the AVP as OctetString..

If there are multiple occurence of AVP(28).

To get the first instance the expression would be: radius.REQ.AVP(28).VALUE.

Returns: radius_avp_value_t

VALUE(uint instanceno)

Returns value of an instance of the AVP.
The first instance of the AVP can be retrieved by simply using the

expression: radius.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

VALUE(0) retrieves the first instance, VALUE(1) retrieves the second

instance, and so on.

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="radius_avp_value_t.html">radius_avp_value_t</a></p></div></div></
div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;

- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by
```

```
      /Book/creator"

      in the first 1000 bytes of the body which is

        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost'
```

```
    }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:

  {
```

```
   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)
```

```
      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

    This will select "all the nodes enclosed by /Book"

    in the first 1000 bytes of the body.

    Since this is a node-set the corresponding string

    value will be returned which is "MiltonParadise
```

```
      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is
```

```
      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
```

```
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

RADIUS_AVP_VALUE_T

Provides operations to typecast RADIUS AVP value to

Integer value or IP Address.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="radius_avp_value_t.html">radius_avp_value_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="radius_avp_value_t.html">radius_avp_value_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
TYPECAST_IPV6_ADDRESS_AT

Typecast the RADIUS AVP value to an IPv6 Address value.

Returns: ipv6_address_at
*** TYPECAST_IPV6_ADDRESS_T**

Convert text_t to ipv6_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_t : This object represents an IPv6 address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
TYPECAST_IPV6_SUBNET_AT

Typecast the RADIUS AVP value to an IPv6 Subnet address value.

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

TYPECAST_IP_ADDRESS_AT

Typecast the RADIUS AVP value to an IP Address value.

Returns: ip_address_at

*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
```

```
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t

TYPECAST_NUM_AT

Typecast the RADIUS AVP value to a Number value.

Returns: num_at

*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Typecast the
RADIUS AVP value to a text value.
```

Note: For known AVP types (mentioned in RFC 2585, 2586) this expression

will identify the value data type and perform typecast to string correctly.

Example: RADIUS.REQ.AVP(8).VALUE.TYPECAST_TEXT_T will identify that AVP(8)

value is Framed IP Address and will return IP Address in dotted decimal format.

However, for AVPs not mentioned in RFC 2585 and 2586 we need to explicitly typecast to ip address or num type.

Example:
RADIUS.REQ.AVP(8).VALUE.TYPECAST_IP_ADDRESS_AT.TYPECAST_TEXT_T.eq("1.1.1.1")

Returns: text_t

TYPECAST_TIME_AT

Typecast the RADIUS AVP value to time value.

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

TYPECAST_UNSIGNED_LONG_AT

Typecast the RADIUS AVP value to 8 octets unsigned long value.

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

> identifies the certkey object containing the RSA

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

RADIUS_CODE_E

This object provides RADIUS message codes.

This topic lists the expressions that are provided by this class.

Enumerations

ACCESS_ACCEPT

ACCESS_CHALLENGE

ACCESS_REJECT

ACCESS_REQUEST

ACCOUNTING_REQUEST

ACCOUNTING_RESPONSE

STATUS_CLIENT

STATUS_SERVER

Expressions

EQ(radius_code_em)

The result is Boolean TRUE if the input argument is equal to

the RADIUS message code by this object.

Parameters(expressions not allowed):

m -

```
RADIUS Message Code
```

Returns: bool_at

NE(radius_code_em)

The result is Boolean FALSE if the input argument is equal to

the RADIUS message code by this object.

Parameters(expressions not allowed):

m -

```
RADIUS Message Code
```

Returns: bool_at

RADIUS_CODE_T

This object provides Num operations and radius_code Type Enum

operations on the RADIUS message.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from radius_code_e, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(radius_code_em)

The result is Boolean TRUE if the input argument is equal to

the RADIUS message code by this object.

Parameters(expressions not allowed):

m -

```
RADIUS Message Code
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GE**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** GT**(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LE**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** LT**(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(radius_code_em)

The result is Boolean FALSE if the input argument is equal to

the RADIUS message code by this object.

Parameters(expressions not allowed):

m -

```
RADIUS Message Code
```

Returns: bool_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** NE**(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert radius_code_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

*** TYPECAST_TEXT_T**

Typecast the Number Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

RADIUS_MESSAGE_T

Define common operations for radius.Req or radius.Res.

This topic lists the expressions that are provided by this class.

ACCT_MULTI_SESSION_ID

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: radius_avp_t

ACCT_SESSION_ID

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: radius_avp_t

ACCT_STATUS_TYPE

Returns the ACCT_STATUS_TYPE AVP (code 40) from the message.

Returns: radius_avp_t

AVP_END

Returns the location at the end of radius message (or in other words

end of all AVPs) in radius message.

Returns: text_t

AVP_LIST

Returns the AVP region in radius message (or in other words

returns location at start of AVPs and length equal to radius

message length excluding header).

Returns: text_t

AVP(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 26 from a Request, use the expression:

radius.REQ.AVP(26).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: radius_avp_t

AVP(uint avpcode, uint vendorid)

Returns the specified AVP encoded within a Vendor-Specific Radius

attribute with the given vendorid from the message.

For example, to get Vendor AVP 1 with vender ID '10415' from a Request, use the expression:

radius.REQ.AVP(10, 10415).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor ID
```

Returns: radius_avp_t

AVP_START

Returns the location at the start of AVPs (or in other words

end of radius header) in radius message.

Returns: text_t

CALLED_STATION_ID

Returns the CALLED_STATION_ID AVP (code 30) from the message.

Returns: radius_avp_t

CALLING_STATION_ID

Returns the CALLING_STATION_ID AVP (code 31) from the message.

Returns: radius_avp_t

CODE

Returns code header field from the message.

Returns: radius_code_t

FRAMED_IP_ADDRESS

Returns the FRAMED_IP_ADDRESS AVP (code 8) from the message.

Returns: radius_avp_t

FRAMED_PROTOCOL

Returns the FRAMED_PROTOCOL AVP (code 7) from the message.

Returns: radius_avp_t

IDENTIFIER

Returns identifier header field from the message.

Returns: num_at

LENGTH

Returns the Message Length from the message.

Returns: num_at

NAS_IDENTIFIER

Returns the NAS_IDENTIFIER AVP (code 32) from the message.

Returns: radius_avp_t

NAS_IP_ADDRESS

Returns the NAS-IP-ADDRESS AVP (code 4) from the message.

Returns: radius_avp_t

NAS_PORT

Returns the NAS_PORT AVP (code 5) from the message.

Returns: radius_avp_t

USER_NAME

Returns the User-Name AVP (code 1) from the message.

Returns: radius_avp_t

VENDOR_SPECIFIC

Returns the Vendor-Id AVP (code 26) from the message.

Returns: vendor_spec_radius_avp_t

RADIUS_PROTOCOL_T

RADIUS Protocol

This topic lists the expressions that are provided by this class.

ATTR_TYPE(uint i)

Returns attribute value for given type

Parameters:

i -

```
Attribute Number
```

Returns: radius_avp_value_t

USERNAME

Returns user name.

Returns: text_t

RADIUS_REQ_T

Retrieves header information and AVPs from radius requests.

AVPs can be retrieved by:

- Directly specifying the AVP codes in the form: radius.REQ.AVP(<avpcode>). For example, to retrieve the NAS-IP-Address AVP (code 4), you can use the expression: radius.REQ.AVP(4).
- Using aliases that are provided for important AVPs. For example, the User-Name AVP has the alias USER_NAME. So, the expression to retrieve the User-Name AVP by using the alias is radius.req.USER_NAME

General usage:radius.REQ.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from radius_message_t.

*** ACCT_MULTI_SESSION_ID**

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: radius_avp_t

*** ACCT_SESSION_ID**

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: radius_avp_t

*** ACCT_STATUS_TYPE**

Returns the ACCT_STATUS_TYPE AVP (code 40) from the message.

Returns: radius_avp_t

*** AVP_END**

Returns the location at the end of radius message (or in other words

end of all AVPs) in radius message.

Returns: text_t

*** AVP_LIST**

Returns the AVP region in radius message (or in other words

returns location at start of AVPs and length equal to radius

message length excluding header).

Returns: text_t

*** AVP**(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 26 from a Request, use the expression:

radius.REQ.AVP(26).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: radius_avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP encoded within a Vendor-Specific Radius

attribute with the given vendorid from the message.

For example, to get Vendor AVP 1 with vender ID '10415' from a Request, use the expression:

radius.REQ.AVP(10, 10415).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor ID
```

Returns: radius_avp_t

*** AVP_START**

Returns the location at the start of AVPs (or in other words

end of radius header) in radius message.

Returns: text_t

*** CALLED_STATION_ID**

Returns the CALLED_STATION_ID AVP (code 30) from the message.

Returns: radius_avp_t

*** CALLING_STATION_ID**

Returns the CALLING_STATION_ID AVP (code 31) from the message.

Returns: radius_avp_t

*** CODE**

Returns code header field from the message.

Returns: radius_code_t

*** FRAMED_IP_ADDRESS**

Returns the FRAMED_IP_ADDRESS AVP (code 8) from the message.

Returns: radius_avp_t

*** FRAMED_PROTOCOL**

Returns the FRAMED_PROTOCOL AVP (code 7) from the message.

Returns: radius_avp_t

*** IDENTIFIER**

Returns identifier header field from the message.

Returns: num_at

*** LENGTH**

Returns the Message Length from the message.

Returns: num_at

*** NAS_IDENTIFIER**

Returns the NAS_IDENTIFIER AVP (code 32) from the message.

Returns: radius_avp_t

*** NAS_IP_ADDRESS**

Returns the NAS-IP-ADDRESS AVP (code 4) from the message.

Returns: radius_avp_t

*** NAS_PORT**

Returns the NAS_PORT AVP (code 5) from the message.

Returns: radius_avp_t

TYPECAST_RADIUS_MESSAGE_T

Convert radius_req_t to radius_message_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_message_t : Define common operations for radius.Req or radius.Res.)

Returns: radius_message_t

*** USER_NAME**

Returns the User-Name AVP (code 1) from the message.

Returns: radius_avp_t

*** VENDOR_SPECIFIC**

Returns the Vendor-Id AVP (code 26) from the message.

Returns: vendor_spec_radius_avp_t

RADIUS_RES_T

Retrieves header information and AVPs from radius responses.

AVPs can be retrieved by:

- Directly specifying the AVP codes in the form: radius.RES.AVP(<avpcode>). For example, to retrieve the PROXY STATE AVP (code 33), you can use the expression: radius.RES.AVP(33).

General usage:radius.RES.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from radius_message_t.

*** ACCT_MULTI_SESSION_ID**

Returns the Acct-Multi-Session-Id AVP (code 50) from the message.

Returns: radius_avp_t

*** ACCT_SESSION_ID**

Returns the Acct-Session-ID AVP (code 44) from the message.

Returns: radius_avp_t

*** ACCT_STATUS_TYPE**

Returns the ACCT_STATUS_TYPE AVP (code 40) from the message.

Returns: radius_avp_t

*** AVP_END**

Returns the location at the end of radius message (or in other words

end of all AVPs) in radius message.

Returns: text_t

*** AVP_LIST**

Returns the AVP region in radius message (or in other words

returns location at start of AVPs and length equal to radius

message length excluding header).

Returns: text_t

*** AVP**(uint avpcode)

Returns the specified AVP from the message.

For example, to get the AVP with code 26 from a Request, use the expression:

radius.REQ.AVP(26).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: radius_avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP encoded within a Vendor-Specific Radius

attribute with the given vendorid from the message.

For example, to get Vendor AVP 1 with vender ID '10415' from a Request, use the expression:

radius.REQ.AVP(10, 10415).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor ID
```

Returns: radius_avp_t

*** AVP_START**

Returns the location at the start of AVPs (or in other words

end of radius header) in radius message.

Returns: text_t

*** CALLED_STATION_ID**

Returns the CALLED_STATION_ID AVP (code 30) from the message.

Returns: radius_avp_t

*** CALLING_STATION_ID**

Returns the CALLING_STATION_ID AVP (code 31) from the message.

Returns: radius_avp_t

*** CODE**

Returns code header field from the message.

Returns: radius_code_t

*** FRAMED_IP_ADDRESS**

Returns the FRAMED_IP_ADDRESS AVP (code 8) from the message.

Returns: radius_avp_t

*** FRAMED_PROTOCOL**

Returns the FRAMED_PROTOCOL AVP (code 7) from the message.

Returns: radius_avp_t

*** IDENTIFIER**

Returns identifier header field from the message.

Returns: num_at

*** LENGTH**

Returns the Message Length from the message.

Returns: num_at

*** NAS_IDENTIFIER**

Returns the NAS_IDENTIFIER AVP (code 32) from the message.

Returns: radius_avp_t

*** NAS_IP_ADDRESS**

Returns the NAS-IP-ADDRESS AVP (code 4) from the message.

Returns: radius_avp_t

*** NAS_PORT**

Returns the NAS_PORT AVP (code 5) from the message.

Returns: radius_avp_t

TYPECAST_RADIUS_MESSAGE_T

Convert radius_res_t to radius_message_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_message_t : Define common operations for radius.Req or radius.Res.)

Returns: radius_message_t

*** USER_NAME**

Returns the User-Name AVP (code 1) from the message.

Returns: radius_avp_t

*** VENDOR_SPECIFIC**

Returns the Vendor-Id AVP (code 26) from the message.

Returns: vendor_spec_radius_avp_t

RADIUS_T

Provides expressions to retrieve radius header information and radius

Header and AVPs (attribute value pairs) from radius requests and responses.

Note: The radius implementation is RFC 2865 and 2866 compliant.

This topic lists the expressions that are provided by this class.

IS_CLIENT

Returns true if this is a radius client connection.

Returns: bool_at

IS_SERVER

Returns true if this is a radius server connection.

Returns: bool_at

NEW_ACCESSREJECT

Creates Radius Access Reject message (Header only) based on Request header.

Returns: text_t

NEW_ANSWER(uint code)

Creates Radius message (Header only) based on Request header.

Parameters:

code -

```
Radius Message code
```

Returns: text_t

NEW_AVP_INTEGER32(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Integer32 value.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given Interger64 value.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, num_atavpvalue)

Creates new AVP based on given avp code and given Interger64 value.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, time_atavpvalue)

Creates new AVP based on given avp code and given Interger64 value.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_INTEGER64(uint avpcode, unsigned_long_atavpvalue)

Creates new AVP based on given avp code and given Interger64 value.

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value
```

Returns: text_t

NEW_AVP_IPADDRESS(uint avpcode, ip_address_atavpvalue)

Creates new AVP based on given avp code and given IP address value.

Example: RADIUS.NEW_AVP_IPADDRESS(8, 1.1.1.1)

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value as octet string
```

Returns: text_t

NEW_AVP_IPV6ADDRESS(uint avpcode, ipv6_address_atavpvalue)

Creates new AVP based on given avp code and given IPv6 address value.

Example: RADIUS.NEW_AVP_IPV6ADDRESS(95, 127:1:1::1)

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value as octet string
```

Returns: text_t

NEW_AVP_IPV6SUBNET(uint avpcode, ipv6_subnet_ats)

Creates new AVP based on given avp code and given IPv6 Subnet value.

Example: RADIUS.NEW_AVP_IPV6SUBNET(97, 127::1/64)

Parameters:

avpcode -

```
AVP code
```

s -

```
IPv6 Subnet
```

Returns: text_t

NEW_AVP(uint avpcode, text_tavpvalue)

Creates new AVP based on given avp code and given value.

Example: RADIUS.NEW_AVP(1, "Hemant")

Example: RADIUS.NEW_AVP(97, RADIUS.REQ.AVP(97).VALUE(0))

Parameters:

avpcode -

```
AVP code
```

avpvalue -

```
AVP value as octet string
```

Returns: text_t

NEW_AVP_TIME(uint avpcode, time_att)

Creates new AVP based on given avp code and given time value.

Example: RADIUS.NEW_AVP(

```
, GMT May 10h 0m)



<b>Parameters</b>:



avpcode -
```

AVP code

t -

Time

Returns: [text_t](text_t.md)

NEW_AVP_UNSIGNED32(uint avpcode, uint avpvalue)

Creates new AVP based on given avp code and given Unsigned32 value.

Parameters:

avpcode -

AVP code

avpvalue -

AVP value

Returns: [text_t](text_t.md)

NEW_AVP_UNSIGNED64(uint avpcode, [ip_address_at](ip_address_at.md)avpvalue)

Creates new AVP based on given avp code and given unsigned64 value.

Parameters:

avpcode -

AVP code

avpvalue -

AVP value

Returns: [text_t](text_t.md)

NEW_AVP_UNSIGNED64(uint avpcode, [num_at](num_at.md)avpvalue)

Creates new AVP based on given avp code and given unsigned64 value.

Parameters:

avpcode -

AVP code

avpvalue -

AVP value

Returns: [text_t](text_t.md)

NEW_AVP_UNSIGNED64(uint avpcode, [time_at](time_at.md)avpvalue)

Creates new AVP based on given avp code and given unsigned64 value.

Parameters:

avpcode -

AVP code

avpvalue -

AVP value

Returns: [text_t](text_t.md)

NEW_AVP_UNSIGNED64(uint avpcode, [unsigned_long_at](unsigned_long_at.md)avpvalue)

Creates new AVP based on given avp code and given unsigned64 value.

Parameters:

avpcode -

AVP code

avpvalue -

AVP value

Returns: [text_t](text_t.md)

NEW_VENDOR_SPEC_AVP(uint vendorid, [text_t](text_t.md)avpvalue)

Creates new Vendor-Specific AVP with given vendor id and given value.

Parameters:

vendorid -

Vendor ID

avpvalue -

AVP value as octet string

Returns: [text_t](text_t.md)

REQ

Retrieves header information and AVPs from radius requests.

Returns: [radius_req_t](radius_req_t.md)

RES

Retrieves header information and AVPs from radius responses.

Returns: [radius_res_t](radius_res_t.md)

SEQUENCE_AT

Finite Sequence Interface

This topic lists the expressions that are provided by this class.

SEQ_TEXT_T

Represents the Text Sequence class.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, sequence_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

> the boolean value to append.

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

> the double value to append.

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

> the IPv4 address to append.

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Returns the number of elements present in the sequence.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET**(uint i)

Returns the element present at position 'i'.

Parameters(expressions not allowed):

i -

```
Number
```

Returns: seq_text_t

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"

    Percent: "%"

    Ampersand: ";"

    Plus: "+"

    Comma: ","

    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
```

```
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="seq_text_t.html">seq_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX, Text operations which attempt to treat the multiple decoded bytes as distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="seq_text_t.html">seq_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
seq_text_t to text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


        The original JSON document is not disturbed but the


        part of the JSON body selected by the


        expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                          XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"


      in the first 1000 bytes of the body. Since this is a node-set


      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SERVER_ATTR_E

Defines the server attributes allowed in DETERMINE_SERVICES().

This topic lists the expressions that are provided by this class.

Enumerations

DESCRIPTION

IP

IPPORT

NAME

PORT

SERVER_T

Designates an expression that refers to the server. This is the starting point for access

into paramters such as Ether and SSL.

This topic lists the expressions that are provided by this class.

ETHER

Operates on the ethernet protocol data associated with the current packet.

Returns: ether_res_protocol_t

INTERFACE

Designates an expression that refers to the ID of the network interface through which

the current packet entered the Application Switch.

Returns: interface_res_t

IP

Operates on the IP protocol data associated with the current packet.

Returns: ip_res_protocol_t

IPV6

Operates on the IPv6 protocol data associated with the current

packet.

Returns: ipv6_res_protocol_t

TCP

Operates on the TCP protocol data associated with the current packet.

Returns: tcp_res_protocol_t

VLAN

Operates on the VLAN through which the current packet entered the Application Switch.

Returns: vlan_res_t

VXLAN

Operates on the VXLAN through which the current packet entered the Application Switch.

Returns: vxlan_res_t

SERVICEGROUP_T

This object represents the servicegroup.

This topic lists the expressions that are provided by this class.

MEMBER(text_tip, uint port)

Returns the service member in the servicegroup

Parameters(expressions not allowed):

ip -

```
IP address
```

port -

```
Port
```

Returns: servicemember_t

NAME

Returns the name of the servicegroup.

Returns: text_t

SERVICEMEMBER_T

This object represents the servicemember in the service group.

This topic lists the expressions that are provided by this class.

CONNECTIONS

Returns number of connections on the service member as an unsigned

long number.

Returns: unsigned_long_at

MONITOR(text_tmonitor_name)

Used for monitor expressions.

These expressions raises an undef if monitor with the given name

doesn't exist.

Parameters:

monitor_name -

```
Monitor name
```

Returns: monitor_t

RESPTIME

Returns as a number, the response time in micro-seconds.

RESPTIME is the TTFB (Time To recieve First Byte) from

all the backend service members.

Returns: num_at

SNMP_TABLE(uint index)

Returns the load entry in the SNMP table for the service member

at the given index.

Parameters(expressions not allowed):

index -

```
Index
```

Raise undef when service member doesnot have SNMP load table or

the index is out of boundary(valid index values are 0-9)

Returns: unsigned_long_at

STATE

Returns the state of service member.

Returns: vserver_state_e

SURGECOUNT

Returns as a number, the number of requests in surge queue of service.

Returns: num_at

THROUGHPUT

Returns throughput of the service in Mbps (Megabits per sec) as

an unsigned long number.

Returns: unsigned_long_at

SERVICEPATH_T

Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns true if the AVP exists for the subscriber.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
    Percent: "%"

    Ampersand: ";"

    Plus: "+"

    Comma: ","

    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

IS_NEXT(text_tsfname)

Returns true if the given service function is the next service function to be taken in the servicepath,

else returns false.

Parameters(expressions not allowed):

sfname -

```
ServiceFunction name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of number type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>IS_NUM_AT</b>(<a
href="num_format_e.html">num_format_e</a>format)</p><div><p>Results in
Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
```

```
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="servicepath_t.html">servicepath_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="servicepath_t.html">servicepath_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.


<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.


Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
servicepath_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton"</
p> Parameters(expressions not allowed): xpathex -

XPath Expression Returns: <a
href="text_t.html">text_t </body></html>

SERVICE_T

Operates on a service

This topic lists the expressions that are provided by this class.

CONNECTIONS

Returns number of connections on the service as an unsigned

long number.

Returns: unsigned_long_at

MONITOR(text_tmonitor_name)

Used for monitor expressions. These expressions raises an undef if

monitor with the given name doesn't exist.

Parameters:

monitor_name -

```
Monitor name
```

Returns: monitor_t

NAME

Returns the name of the service.

Returns: text_t

RESPTIME

Returns as a number, the response time in micro-seconds.

RESPTIME is the TTFB (Time To recieve First Byte) from

all the backend services.

Returns: num_at

SNMP_TABLE(uint index)

Returns the load entry in the SNMP table at the given index for

the service.

Parameters(expressions not allowed):

index -

```
Index
```

Raises an undef when service doesnot have SNMP load table or

the given index is out of boundary (valid index values are 0-9).

Returns: unsigned_long_at

STATE

Returns the state of service.

Returns: vserver_state_e

SURGECOUNT

Returns as a number, the number of requests in surge queue of service.

Returns: num_at

THROUGHPUT

Returns throughput of the service in Mbps (Megabits per sec) as

an unsigned long number.

Returns: unsigned_long_at

SERVICE_TYPE_E

Connection service type

This topic lists the expressions that are provided by this class.

Enumerations

ADNS

ANY

DHCPRA

FIX

FTP

MONITOR

MONITOR_PING

MONITOR_UDP

NAT

NNTP

PKTSTEER

RPCCLNT

RPCSVR

RPCSVRS

RTSP

SERVICE_UNKNOWN

SIP_TCP

SIP_TLS

SIP_UDP

SNMP

SSL

SSL_BRIDGE

SSL_FIX

SSL_TCP

SVC_AAA

SVC_DNS

SVC_HTTP

SVC_MSSQL

SVC_MYSQL

TCP

UDP

Expressions

EQ(service_type_etype)

Returns Boolean True if the Connection service type matches the argument.

Example: CONNECTION.SERVICE_TYPE.EQ(TCP)

Parameters(expressions not allowed):

type -

```
Service type
```

Returns: bool_at

NE(service_type_etype)

Returns Boolean True if the Connection service type do not matches the argument.

Example: CONNECTION.SERVICE_TYPE.NE(TCP)

Parameters(expressions not allowed):

type -

```
Service type
```

Returns: bool_at

SIP_ADDRESS_AT

This object provides operations for parsing the SIP To header.

This topic lists the expressions that are provided by this class.

SIP_CALLID_HEADER_T

This object provides operations for parsing the SIP callid header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sip_header_t.

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LIST</b></p><div><p>A SIP
header can have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_callid_header_t.html">sip_callid_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

ascii value of character Y in hexadecimal.

For example, the encoded text "http%3a//" will be decoded to "http://"

2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the

ascii value of character Y in hexadecimal.

For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_callid_header_t.html">sip_callid_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
```

```
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T

Convert sip_callid_header_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : Typecast an SIP Header object of a specific type to SIP

Header.For example: SIP.REQ.CONTENT_LENGTH.TYPECAST_SIP_HEADER_T)

Returns: sip_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_header_t to text_t. Will raise an undef if the value is null, empty,
```

```
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
```

```
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id

- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
```

```
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
       "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_CONTENT_LENGTH_T

The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sip_header_t, num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

number.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

> the time value to append.

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** INSTANCE**(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at

*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LIST</b></p><div><p>A SIP
header can have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the minimum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
```

```
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_content_length_t.html">sip_content_length_t</a></p></div></
div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX, Text operations which attempt to treat the multiple decoded bytes as distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="sip_content_length_t.html">sip_content_length_t</a></p></div></
div></div><div><div><p><b>* </b><b>SIGNED16_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
16-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t
*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```


treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.
```

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should

produce the output "ABCD:1234::ABCD10.100.10.100".

Similarly, CLIENT.IPV6.SRC + 5 should produce the result

ABCD:1234::ABCD5) Returns: ipv6_address_at </

```
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t

*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
TYPECAST_NUM_AT

Convert sip_content_length_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T

Convert sip_content_length_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : Typecast an SIP Header object of a specific type to SIP

Header.For example: SIP.REQ.CONTENT_LENGTH.TYPECAST_SIP_HEADER_T)

Returns: sip_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

destination type. (tcp_options_t : TCP Protocol Options) Returns:
tcp_options_t </
div> * TYPECAST_TCP_OPTIONS_TYPE_T</
p> Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p> Returns: <a
href="tcp_options_type_t.html">tcp_options_type_t </
div> * TYPECAST_TCP_REPEATER_OPTION_T</
p> Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater) Returns: <a
href="tcp_repeater_option_t.html">tcp_repeater_option_t </
div> * TYPECAST_TEXT_T Typecast
the Number Value to its string representation. Returns: <a
href="text_t.html">text_t * </
b>TYPECAST_TIME_AT Typecast the Number Value to a Time
value. Returns: time_at </
div> * TYPECAST_TIME_T</
p> Convert text_t to time_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>WEEKDAY_STRING</b></
p><div><p>Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,
```

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
```

```
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)
```

```
  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)



      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
           "person":{


             "name":'Milton'


           }


       },


       "title":'Paradise Lost',


       "price":"36"


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


     This will convert the string '36' string to double if it is


     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
```

```
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)
```

```
      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_FROM_HEADER_T

This object provides operations for parsing the SIP From header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sip_header_t, sip_address_at.

*** ADDRESS**

Returns the SIP URI in the "To" or "From" header

Returns: sip_url_t

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DISPLAY_NAME**

Returns the display name in the "To" or "From" header

Returns: sip_url_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LIST</b></p><div><p>A SIP
header can have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t
*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_from_header_t.html">sip_from_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_from_header_t.html">sip_from_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TAG</b></
p><div><p>Returns the "tag" parameter in the "To" or "From" header</
p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_LOWER</b></p><div><p>Converts the selected
text to lower case.<br /><br /><br /><br />For example, the string "ABCd"
is converted to "abcd"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TO_UPPER</b></p><div><p>Converts the selected text to upper
case.<br /><br /><br /><br />For example, the string "abcD" is converted
to "ABCD"</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>TRUNCATE</b>(char c, uint i)</
p><div><p>Select the Text after truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T

Convert sip_from_header_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : Typecast an SIP Header object of a specific type to SIP

Header.For example: SIP.REQ.CONTENT_LENGTH.TYPECAST_SIP_HEADER_T)

Returns: sip_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_HEADER_T

The SIP.REQ.HEADER("header name") or SIP.RES.HEADER("header name")

object represents the multiple lines of a particular SIP header

type. This object is also a Text object. Direct Text operations

on this object will mostly operate only on the value in the last header

line of this type. Some Text operations like CONTAINS have been

overridden to operate on values in all the header lines of this

type.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

AFTER_REGEX(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

AFTER_STR(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

BEFORE_REGEX(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

BEFORE_STR(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

CONTAINS_ANY(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

CONTAINS_INDEX(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

CONTAINS(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

COUNT

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

EQUALS_ANY(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

EQUALS_INDEX(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

INSTANCE(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>LIST</b></p><div><p>A SIP header can
have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>REGEX_MATCH</b>(re regex)</
p><div><p>The REGEX_MATCH operator in this object overrides the Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()'" at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT
operator in this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>The SUBSTR operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.


<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.


Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

VALUE(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex -

XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_HOSTNAME_T

This object provides operations for parsing the SIP hostname as it

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

DOMAIN

Operate as Text on the Domain name part of the hostname.

If the hostname is www.foobar.com or www.foobar.com:5060, then

domain is foobar.com. This operation will return incorrect

results if the hostname has an IP address. This method also sets

the text mode to case-insensitive. So all text operations after

this method will operate in a case-insensitive manner.

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

EQ(text_thostname)

Operates on the text contents of hostname. Results in Boolean TRUE

value if the hostname matches the input argument.

The comparison will be case-insensitive and if textmode is URLENCODED

the hostname will be decoded before comparison.

If hostname is www.foobar.com or www.foobar.com., then

EQ(www.forbar.com) returns TRUE.

Parameters:

hostname -

```
Hostname
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

Based on RFC2396, following are considered URL safe characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

```
Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at

*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

PORT

Operate as Number on the port part of the hostname. The

string following and including the first : is considered the

port value.

Examples:

If the hostname is www.foo.com:5060 then the port

is ":5060".

If the hostname is www.foo.com: then the port is

":".

If the hostname is www.foo.com then the port is "" and

points to just after ".com".

When the numerical value in the port is missing then it

assumes a default value of 5060.

Returns: num_t

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SERVER</b></p><div><p>Operate as Text on the Server
name part of the hostname.
```

If the hostname is www.foobar.com or www.foobar.com:5060, then

server is www.foobar.com. This method also sets the text mode to

case-insensitive. So all text operations after this method

will operate in a case-insensitive manner.

Returns: text_t
*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_hostname_t.html">sip_hostname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_hostname_t.html">sip_hostname_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
```

```
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```


supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_hostname_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{


   "Book":{


      "creator":{


         "person":{


           "name":'Milton'


         }


      },


      "title":'Paradise Lost'


   }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"


      in the first 1000 bytes of the body. Since this is a node-set


      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_METHOD_E

This object represents one of the SIP Request Methods.

This topic lists the expressions that are provided by this class.

Enumerations

ACK

BYE

CANCEL

INVITE

OPTIONS

REGISTER

UNKNOWN_METHOD

Expressions

EQ(sip_method_em)

The result is Boolean TRUE if the input argument is equal to

the SIP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
SIP Request Method Name
```

Returns: bool_at

NE(sip_method_em)

The result is Boolean TRUE if the input argument is not equal to

the SIP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
SIP Request Method Name
```

Returns: bool_at

SIP_METHOD_T

This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, sip_method_e.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(sip_method_em)

The result is Boolean TRUE if the input argument is equal to

the SIP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
SIP Request Method Name
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
```

```
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."
```

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

```
    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'

- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="sip_method_e.html">sip_method_e</a>m)</p><div><p>The result is
Boolean TRUE if the input argument is not equal to
```

the SIP Request Method represented by this object.

Parameters(expressions not allowed):

m -

```
SIP Request Method Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>NE</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Results in
Boolean TRUE value if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_method_t.html">sip_method_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_method_t.html">sip_method_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_method_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"


      in the first 1000 bytes of the body. Since this is a node-set


      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_PROTOCOL_T

This is the starting point for SIP protocol based expressions

This topic lists the expressions that are provided by this class.

REQ

Operates on SIP Request.

Returns: sip_req_t

RES

Operates on SIP Response.

Returns: sip_res_t

SIP_REQ_T

Operates on SIP Request.

This topic lists the expressions that are provided by this class.

BODY(uint i)

Operate as Text on the SIP request body. The length of the

body is equal to the number in the argument. If there aren't

enough characters in the body then the entire body is

selected.

Parameters:

i -

```
Body length
```

Returns: text_t

CALLID

Returns the content of the "Callid" header

Returns: sip_callid_header_t

CONTENT_LENGTH

Returns the length of the SIP request as a number.

Returns: sip_content_length_t

CSEQ

Returns the SIP Cseq number

Returns: num_t

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

FROM

Returns the content of the "From" header

Returns: sip_from_header_t

FULL_HEADER

Returns the entire request header (including the

terminating \r\n\r\n)

Returns: text_t

HEADER(text_theader_name)

Returns the contents of the SIP header specified by the String argument.

The header name cannot be longer than 32 characters.

Parameters(expressions not allowed):

header_name -

```
Header Name
```

Returns: sip_header_t

IS_VALID

Returns TRUE if the SIP request is properly formed.

Returns: bool_at

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

METHOD

Returns the SIP method object from the request.

Returns: sip_method_t

TO

Returns the content of the "To" header

Returns: sip_to_header_t

URL

Obtain the SIP URL object from the request. The Text object

backing the SIP URL object has the Text Mode set to

URLENCODED by default.

Returns: sip_url_t

VERSION

Returns the SIP version.

Returns: sip_version_t

VIA

Returns the content of the "Via" header

Returns: sip_via_header_t

SIP_RES_T

Operates SIP Response.

This topic lists the expressions that are provided by this class.

BODY(uint i)

Operate as Text on the SIP response body. The length of the body is equal

to the number in the argument. If there aren't enough characters in the

body then the entire body is selected.

Parameters:

i -

```
Body length
```

Returns: text_t

CALLID

Returns the content of the "Callid" header

Returns: text_t

CONTENT_LENGTH

Returns the length of the SIP response as a number.

Returns: sip_content_length_t

CSEQ

Returns the SIP Cseq number

Returns: num_t

FROM

Returns the content of the "From" header

Returns: sip_from_header_t

FULL_HEADER

Returns the entire response header (including the

terminating \r\n\r\n)

Returns: text_t

HEADER(text_theader_name)

Returns the contents of the SIP header specified by the String argument.

The header name cannot be longer than 32 characters.

Parameters(expressions not allowed):

header_name -

```
Header Name
```

Returns: sip_header_t

IS_GLOBAL_FAILURE

Returns True if response code is 6xx.

Returns: bool_at

IS_PROVISIONAL

Returns True if response code is 1xx.

Returns: bool_at

IS_REDIRECT

Returns True if response code is 3xx

Returns: bool_at

IS_REQUEST_FAILURE

Returns True if response code is 4xx.

Returns: bool_at

IS_SERVER_FAILURE

Returns True if response code is a 5xx.

Returns: bool_at

IS_SUCCESSFUL

Returns True if response code is 2xx.

Returns: bool_at

IS_VALID

Returns TRUE if the SIP response is properly formed.

Returns: bool_at

METHOD

Returns the SIP method. this is extracted from the Cseq header in the response.

Returns: sip_method_t

STATUS

Returns the SIP response status code.

Returns: num_t

STATUS_MSG

Returns the SIP response status message.

Returns: text_t

TO

Returns the content of the "To" header

Returns: sip_to_header_t

VERSION

Returns the SIP version.

Returns: sip_version_t

VIA

Returns the content of the "Via" header

Returns: sip_via_header_t

SIP_TO_HEADER_T

This object provides operations for parsing the SIP From header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sip_header_t, sip_address_at.

*** ADDRESS**

Returns the SIP URI in the "To" or "From" header

Returns: sip_url_t

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DISPLAY_NAME**

Returns the display name in the "To" or "From" header

Returns: sip_url_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** INSTANCE**(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LIST</b></p><div><p>A SIP
header can have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t
*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>The REGEX_MATCH operator in this object overrides the
Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_to_header_t.html">sip_to_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_to_header_t.html">sip_to_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TAG</b></
p><div><p>Returns the "tag" parameter in the "To" or "From" header</
p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_LOWER</b></p><div><p>Converts the selected
text to lower case.<br /><br /><br /><br />For example, the string "ABCd"
is converted to "abcd"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TO_UPPER</b></p><div><p>Converts the selected text to upper
case.<br /><br /><br /><br />For example, the string "abcD" is converted
to "ABCD"</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>TRUNCATE</b>(char c, uint i)</
p><div><p>Select the Text after truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T

Convert sip_to_header_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : Typecast an SIP Header object of a specific type to SIP

Header.For example: SIP.REQ.CONTENT_LENGTH.TYPECAST_SIP_HEADER_T)

Returns: sip_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


       The original JSON document is not disturbed but the


       part of the JSON body selected by the


       expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                          XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'



           }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing.</p><p>Parameters(expressions not allowed):</p><div><p>xpathex - XPath Expression</p></div><p>Returns: text_t</p></div></div></div><div><div><p>* XPATH(xpath_bool xpathex)</p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_URL_T

SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header. The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>HOSTNAME</b></p><div><p>Operate as SIP Host Name on
the hostname present in the
```

URL.

It can not be used in bi-directional policies.

Returns: sip_hostname_t

*** HTML_XML_SAFE**

Apply XML safe transform and select text.

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
```

```
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>PROTOCOL</b></p><div><p>Operate as
Text on the protocol present in the URL.
```

It can not be used in bi-directional policies.

Returns: text_t
*** REGEX_MATCH**(re regex)

Returns TRUE if and only if the text contains at least one match

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_url_t.html">sip_url_t</a></p></div></div></div><div><div><p><b>*
</b><b>SET_TEXT_MODE</b>(<a href="textmode_e.html">textmode_e</a>mode)</
p><div><p>The mode setting of a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="sip_url_t.html">sip_url_t</
a></p></div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint
i)</p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert sip_url_t
to text_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (text_t : Designates any
piece of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

USERNAME

Operate as Text on the username present in the SIP URI

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child
```

```
  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is

    in the first 1000 bytes of the document.</p><p><b>Parameters</
```

```
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

SIP_VERSION_T

SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

> the IPv6 subnet to append.

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>MAJOR</b></p><div><p>Operate as Number
on the major component of the SIP Version
```

string.

Returns: num_t
*** MAP_STRING_DEFAULT_TO_KEY**(text_tstring_map_name)

Returns the value corresponding to the key in the

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>MINOR</b></p><div><p>Operate as Number on
the minor component of the SIP Version
```

string.

Returns: num_t
*** MIN**(text_ts)

Compares the string argument with the current string and

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_version_t.html">sip_version_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_version_t.html">sip_version_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_version_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:

    Alpha-numeric characters: a-z, A-Z, 0-9

    Hyphen and underscore: "-", "_"

    Dot: "."

    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's
```

```
  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SIP_VIA_HEADER_T

This object provides operations for parsing the SIP Via header.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from sip_header_t.

*** AFTER_REGEX**(re regex)

The AFTER_REGEX operator in this object overrides the Text

AFTER_REGEX operator to operate on all the instances of the current

header type. Selects the Text after the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select "DAPPER" from
"KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

The AFTER_STR operator in this object overrides the Text AFTER_STR

operator to operate on all the instances of the current header type.

Selects Text after the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: 111abc\r\n Content-Length: 200\r\n MyHdr: 111def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").AFTER_STR("111") selects "def" in the

last instance of MyHdr, the one with value "111def".
SIP.RES.HEADER("MyHdr").AFTER_STR("1ab") selects "c" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

The BEFORE_REGEX operator in this object overrides the Text

BEFORE_REGEX operator to operate on all the instances of the current

header type. Selects the Text before the text that matches the input

regular expression in any instance of the header value. The order in

which the header instances will be matched is from the last to the

first.

For example:

```
SIP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text" from
"text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

The BEFORE_STR operator in this object overrides the Text BEFORE_STR

operator to operate on all the instances of the current header type.

Selects Text before the first occurrence of the input string

argument. The order in which the instances of the header are matched

is from the last instance to the first.

SIP/2.0 200 OK\r\n MyHdr: abc111\r\n Content-Length: 200\r\n MyHdr: def111\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").BEFORE_STR("111") selects "def" in the

last instance of MyHdr, the one with value "def111".
SIP.RES.HEADER("MyHdr").BEFORE_STR("bc1") selects "a" in the first

instance of MyHdr, the one with value "abc111".

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

BRANCH

Returns the BRANCH used for SIP communication

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatclass_name)

The CONTAINS_ANY in this object overrides the Text CONTAINS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patclass is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatclass_name)

The CONTAINS_INDEX in this object overrides the Text CONTAINS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patclass that is present in any instance of the header value.

Parameters(expressions not allowed):

patclass_name -

```
Patclass Name
```

Returns: num_at

*** CONTAINS**(text_ts)

The CONTAINS in this object overrides the Text CONTAINS

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input Pattern

String is present in any instance of the header value.

GET / SIP/2.0\r\n Host: abc.com\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

Examples: SIP.RES.HEADER("MyHdr").CONTAINS("de") is TRUE.
SIP.RES.HEADER("MyHdr").CONTAINS("bcd") is

FALSE. SIP.REQ.HEADER("MyHdr").CONTAINS("de") is

TRUE. SIP.REQ.HEADER("MyHdr").CONTAINS("bcd") is FALSE. The

system does not concatenate the different values.

Parameters(expressions not allowed):

s -

```
Pattern String
```

Returns: bool_at

*** COUNT**

An SIP header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

SIP/2.0 200 OK\r\n MyHdr: abc\r\n Content-Length: 200\r\n MyHdr: def\r\n \r\n

For example: SIP.RES.HEADER("MyHdr").COUNT will return Number 2. The

system does not keep track of more than 15 SIP headers. The result

will be undefined if there are more than 15 instances of the header.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

The EQUALS_ANY in this object overrides the Text EQUALS_ANY

operator to operate on all the instances of the current

header type. Returns Boolean TRUE if any of the patterns associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

The EQUALS_INDEX in this object overrides the Text EQUALS_INDEX

operator to operate on all the instances of the current

header type. Returns the index of the matching pattern associated

with input patset is equal to any instance of the header value.

Parameters(expressions not allowed):

patset_name -

```
Patset Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns TRUE if an instance of the current header type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint lineno)

An SIP header can have multiple occurrences. INSTANCE(0) will

select the last header line of the current type, INSTANCE(1) will

select the last but one header line and so on. The counting goes

backward. Only the last 15 instances of a header are accessible.

Therefore Instance Number cannot be greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").INSTANCE(1) will return a Text

object referring to "MyHdr: abc\r\n".

It can not be used in bi-directional policies.

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>IS_BOOL_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of boolean type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of double data type.</p><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LIST</b></p><div><p>A SIP
header can have multiple comma separated parameter list in a single line.
```

LIST operation will give a list of comma separated entities. LIST.GET(0) will select

the first entity in the comma separated header line. LIST.GET(1) will select the

second entity and so on.

SIP/2.0 200 OK\r\n

MyHdr: abc;param1=1;param2=2,abc2=second;paramabc2=3\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example : SIP.REQ.HEADER("MyHdr").LIST.GET(1) will return a text object referring

to "abc2=second;paramabc2=3"

Returns: list_t

*** LOCATION_LONG**

Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>RECEIVED</b></p><div><p>Returns
the source IP address from the which the end host received the SIP
request.</p><p><b>Returns: </b><a
href="ip_address_at.html">ip_address_at</a></p></div></div></
div><div><div><p><b>RECEIVED_IPV6</b></p><div><p>Returns the source IPV6
address from the which the end host received the SIP request.</
p><p><b>Returns: </b><a href="ipv6_address_at.html">ipv6_address_at</a></
p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re regex)</
p><div><p>The REGEX_MATCH operator in this object overrides the Text
```

REGEX_MATCH operator to operate on all the instances of the current

header type. Returns Boolean TRUE if the input regular expression

matches any instance of the header value. The order in which the

header instances will be matched is from the last to the first.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at sip://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended to not use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>The REGEX_SELECT operator in
this object overrides the Text
```

REGEX_SELECT operator to operate on all the instances of the current

header type. Selects the Text that matches the input regular

expression in any instance of the header value. The order in which

the header instances will be matched is from the last to the first.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>RPORT</b></p><div><p>Returns the RPORT used for SIP
communication</p><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>SENTBY_ADDRESS</b></p><div><p>Returns the
IP address of the end host which sent the SIP request.</p><p><b>Returns:
</b><a href="ip_address_at.html">ip_address_at</a></p></div></div></
div><div><div><p><b>SENTBY_ADDRESS_IPV6</b></p><div><p>Returns the IPV6
address of the end host which sent the SIP request.</p><p><b>Returns: </
b><a href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>SENTBY_PORT</b></p><div><p>Returns the source port of
the end host which send the SIP request.</p><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</a>charset)</
p><div><p>Set the character set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sip_via_header_t.html">sip_via_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
```

```
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sip_via_header_t.html">sip_via_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>The SUBSTR
operator in this object overrides the Text SUBSTR
```

operator to operate on all the instances of the current header type.

Selects Text that matches the input string argument. The order in

which the instances of the header are matched is from the last

instance to the first.

SIP/2.0 200 OK\r\n

MyHdr: 111abc\r\n

Content-Length: 200\r\n

MyHdr: 111def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").SUBSTR("111") selects "111" in the

last instance of MyHdr, the one with value "111def".

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

(http_cache_control_t : This object provides for the ability to work on HTTP Cache-Control headers.) Returns: http_cache_control_t * TYPECAST_HTTP_CONTENT_LENGTH_T Convert text_t to http_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or HTTP.RES.CONTENT_LENGTH object can be

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.


<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.


Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T

Convert sip_via_header_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : Typecast an SIP Header object of a specific type to SIP

Header.For example: SIP.REQ.CONTENT_LENGTH.TYPECAST_SIP_HEADER_T)

Returns: sip_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
```

```
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sip_header_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```


1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**(uint lineno)

An SIP header can have multiple occurrences. VALUE(0) will

select the value in the last header line of the current type,

VALUE(1) will select the value in the last but one header line

and so on. The counting goes backward. Only the last 15 instances

of a header are accessible. Therefore Instance Number cannot be

greater than 14.

SIP/2.0 200 OK\r\n

MyHdr: abc\r\n

Content-Length: 200\r\n

MyHdr: def\r\n

\r\n

For example: SIP.RES.HEADER("MyHdr").VALUE(1) will return a Text

object referring to "abc\r\n".

Parameters(expressions not allowed):

lineno -

```
Instance Number</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>XML_DECRYPT</b>(<a
href="text_t.html">text_t</a>certKeyName)</p><div><p>Decrypt the selected
content, which was encrypted using the W3C XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:


{


   "Book":{


      "creator":{


         "person":{


           "name":'Milton'


         }


      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{



            "name":'Milton'



          }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SMPP_BINDINFO_T

This object operates on bindinfo message of the SMPP Connection.

This topic lists the expressions that are provided by this class.

ADDRESS_RANGE

Returns the ESME address or range of the ESME adresses.

Returns: text_t

ADDR_NPI

Returns the Numbering Plan Indicator (NPI) for the ESME address.

Returns: num_at

ADDR_TON

Returns the Type of Number (ToN) of the ESME address.

Returns: num_at

SYSTEM_ID

Returns the ESME system requesting to bind with the Message Center.

Returns: text_t

SYSTEM_TYPE

Returns the ESME system requesting to bind with the Message Center.

Returns: text_t

TYPE

Enum for the type of bind command. The different values of the

enum will be TRANSCEIVER/TRANSMITTER/RECEIVER.

Returns: smpp_bind_e

SMPP_BIND_E

This object represents one of the Bind Type.

This topic lists the expressions that are provided by this class.

Enumerations

RECEIVER

TRANSCEIVER

TRANSMITTER

Expressions

EQ(smpp_bind_em)

Result is boolean TRUE if input argument is equal to

bind type of this connection.

Parameters(expressions not allowed):

m -

```
SMPP Bind Type
```

Returns: bool_at

NE(smpp_bind_em)

Result is boolean TRUE if input argument is equal to

bind type of this connection.

Parameters(expressions not allowed):

m -

```
SMPP Bind Type
```

Returns: bool_at

SMPP_PROTOCOL_T

Models the SMPP protocol.

This topic lists the expressions that are provided by this class.

BINDINFO

Operates on SMPP BINDINFO.

BINDINFO contains the parameters used by client while doing bind.

Returns: smpp_bindinfo_t

SNI_TEXT_AT

Sni Text Interface

This topic lists the expressions that are provided by this class.

SNMP_TABLE_T

The SYS.VSERVER("vserver name").SNMP_TABLE object represents the snmp load table on the vserver. This table contains the load value aggregated across all the services, servicegroup items bound to the vserver. The load values themselves are retrieved via the snmp load monitor bound to the services and servicegroup items.

This topic lists the expressions that are provided by this class.

AVERAGE_VALUE

Returns the average value of the load parameter at the index.

Returns: unsigned_long_at

SQL_QUERY_T

Describes a SQL query.

This topic lists the expressions that are provided by this class.

SSL_TEXT_T

The client.ssl.client_hello.sni object represents sni.

This object is of type text_t.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, sni_text_at.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t

*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +
```

```
"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="ssl_text_t.html">ssl_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"

2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="ssl_text_t.html">ssl_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>SNI_CERT</b></
p><div><p>Operate as SNI on the Ssl Text component.
```

Returns the SSL certificate based on the text provided.

Returns: x509_certificate_t
*** STARTSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text starts with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).

- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t
*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL
```

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

Example: Let the client source IPv6 address be ABCD:1234::ABCD.

Let the server destination IPv4 address be 10.100.10.100.

```
Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t

*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
```

```
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t
*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
ssl_text_t to text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
```

```
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
```

```
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the

Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
```

```
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9
```

```
Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
   HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



   returns the &lt;title&gt; element in the document's



   &lt;head&gt; This can be used in a rewrite action



   to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)


  returns true if the &lt;meta&gt; element exists as a child


  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)


  returns the numeric value of the height attribute of the
```

```
  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }

    The original JSON document is not disturbed but the
```

```
      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{


            "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                          XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
```

```
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```


returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:


```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

SSLVPN_MODE_E

This object represents one of the SSLVPN Modes.

This topic lists the expressions that are provided by this class.

Enumerations

CLIENTLESS_ENC_MODE

CLIENTLESS_OPAQUE_MODE

CLIENTLESS_TRANSPARENT_MODE

ICA_PROXY_MODE

SECURE_CLIENT_MODE

UNDEFINED_MODE

Expressions

EQ(sslvpn_mode_em)

The result is Boolean TRUE if the input argument is equal to

the SSLVPN Mode enum represented by this object.This is not

used currently since there is no way of passing an enum object

from PI expression for comparision.

Parameters(expressions not allowed):

m -

```
SSLVPN mode enum
```

Returns: bool_at

SSLVPN_MODE_T

This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, sslvpn_mode_e.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(sslvpn_mode_em)

The result is Boolean TRUE if the input argument is equal to

the SSLVPN Mode enum represented by this object.This is not

used currently since there is no way of passing an enum object

from PI expression for comparision.

Parameters(expressions not allowed):

m -

```
SSLVPN mode enum
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
```

```
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="sslvpn_mode_t.html">sslvpn_mode_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace. For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating on bytes). By default a single byte represents a character using the ASCII encoding. But if URLENCODED mode is set then 3 bytes may represent a character. A PREFIX(3) operation will select the first 3 characters in the Text object. That means it may select up to 9 bytes for URLENCODED Text object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX, Text operations which attempt to treat the multiple decoded bytes as distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="sslvpn_mode_t.html">sslvpn_mode_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
```

```
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
sslvpn_mode_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


        The original JSON document is not disturbed but the


        part of the JSON body selected by the


        expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                          XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

```
      "all the nodes enclosed by /Book/creator"


      in the first 1000 bytes of the body. Since this is a node-set


      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

SSLVPN_REQ_PROTOCOL_T

SSLVPN Protocol

This topic lists the expressions that are provided by this class.

IS_SSLVPN

Returns TRUE if the current connection is SSLVPN-based.

Returns: bool_at

MODE

Returns sslvpn mode + URL Encoding

Valid values are :

"SECURE_ACCESS_CLIENT" -> Mode = Vpn Client

"CVPN_OPAQUE" -> Mode = ClientlessVPN Url Encoding = Opaque

"CVPN_ENCRYPT" -> Mode = ClientlessVPN Url Encoding = Encrypt

"CVPN_TRANSPARENT" -> Mode = ClientlessVPN Url Encoding = Transparent

"ICAPROXY" -> Mode = ICA

Returns: sslvpn_mode_t

SSL_REQ_PROTOCOL_T

SSL Protocol

This topic lists the expressions that are provided by this class.

CIPHER_BITS

Returns the number of bits in the cryptographic key. Returns 0 if the transaction is not SSL based.

Returns: num_at

CIPHER_EXPORTABLE

Returns TRUE if the SSL cryptographic key is exportable.

Returns: bool_at

CIPHER_NAME

Returns SSL Cipher name. Returns a NULL string if invoked from a non SSL connnection

Returns: text_t

CLIENT_CERT

Returns the SSL client certificate in the current SSL transaction.

Returns: x509_certificate_t

CLIENT_HELLO

Returns SSL Client Hello fields.

Returns: client_hello_t

DETECTED_DOMAIN

Returns SSL detected domain. Returns a NULL string if invoked from a non SSL connnection

Returns: text_t

IS_SSL

Returns TRUE if the current connection is SSL-based.

Returns: bool_at

ORIGIN_SERVER_CERT

Returns the certificate of Origin Server in the current SSL transaction.

Returns: x509_certificate_t

VERSION

Returns the SSL protocol version. Returns 0 if the

transaction is not SSL based. Returns 0x002 if the

transaction is SSLv2. Returns 0x300 if the transaction is

SSLv3. Returns 0x301 if the transaction is TLSv1.

Returns: num_at

STREAM_ACTION_E

This object provides stream analytics action types.

This topic lists the expressions that are provided by this class.

Enumerations

DROP

RESET

STREAM_MODE_E

This object provides stream analytics check limit modes.

This topic lists the expressions that are provided by this class.

Enumerations

BANDWIDTH

CONNECTIONS

REQUEST_RATE

RESPONSE_TIME

STREAM_T

Stream

This topic lists the expressions that are provided by this class.

BANDWIDTH

Returns the bandwidth on an object.

Returns: unsigned_long_at

CHECK_BREACH_THRESHOLD(stream_transaction_track_mode_emode)

Returns true if an object corresponding to the stream identifier

has hit maximum breaching transactions threshold.

Transaction Tracking needs to be enabled in stream identifier configuration to use this expression.

Transaction Tracking mode: RESPONSE TIME in milli seconds.

```
add stream identifier sid -interval 5 -tracktransactions RESPTIME
-maxTransactionThreshold 100 -BreachThreshold 10



 Expression: ANALYTIC.STREAM("sid").CHECK_BREACH_THRESHOLD(RESPONSE_TIME)
```

Parameters(expressions not allowed):

mode -

```
CHECK BREACH mode
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, ip_address_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(BANDWIDTH, 100000)
```

Note: maxthreshold should always be greater than 0.

Parameters(expressions not allowed):

mode -

```
Check Limit mode
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, num_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(BANDWIDTH, 100000)
```

Note: maxthreshold should always be greater than 0.

Parameters(expressions not allowed):

mode -

```
Check Limit mode
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, time_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(BANDWIDTH, 100000)
```

Note: maxthreshold should always be greater than 0.

Parameters(expressions not allowed):

mode -

```
Check Limit mode
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, unsigned_long_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(BANDWIDTH, 100000)
```

Note: maxthreshold should always be greater than 0.

Parameters(expressions not allowed):

mode -

```
Check Limit mode
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, ip_address_atminthreshold, ip_address_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, ip_address_atminthreshold, num_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, ip_address_atminthreshold, time_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, ip_address_atminthreshold, unsigned_long_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, num_atminthreshold, ip_address_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5


 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, num_atminthreshold, num_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, num_atminthreshold, time_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, num_atminthreshold, unsigned_long_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, time_atminthreshold, ip_address_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, time_atminthreshold, num_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, time_atminthreshold, time_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, time_atminthreshold, unsigned_long_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, unsigned_long_atminthreshold, ip_address_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5
```

```
Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, unsigned_long_atminthreshold, num_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, unsigned_long_atminthreshold, time_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

CHECK_LIMIT(stream_mode_emode, unsigned_long_atminthreshold, unsigned_long_atmaxthreshold)

Returns true if an object corresponding to the stream identifier

has hit the limit parameter.

mode: BANDWIDTH in kbps, RESPONSE TIME in milli seconds, REQUEST RATE in Requests/second,

CONNECTIONS i.e. Total Active Connections.

```
add stream identifier sid -interval 5



 Expression: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 1000)
```

Note: maxthreshold and minthreshold cannot be negative. maxthreshold cannot be less than minthreshold.

Note: If maxthreshold is 0, then max threshold is not applied. minthreshold and maxthreshold cannot both be zero.

```
Example: ANALYTIC.STREAM("sid").check_limit(REQUEST_RATE, 10, 0)
```

Above expression returns true only if Request Rate is less than 10 requests per second. There is no upper limit here.

Parameters(expressions not allowed):

mode -

```
Check Limit Mode
```

minthreshold -

```
Check Limit min threshold
```

maxthreshold -

```
Check Limit maximum threshold
```

Returns: bool_at

COLLECT_STATS

Returns true if statistics collection on an object was successful.

Returns: bool_at

COLLECT_STATS(uint amplification_factor)

Returns true if statistics collection on an object was successful.

Parameters(expressions not allowed):

amplification_factor -

```
Amplification factor
```

Returns: bool_at

COLLECT_STATS(text_tmode, uint max_threshold, stream_action_eaction)

Returns true if statistics collection on an object was

successful.

This expression is used to rate limit connections per packet

level. When an object/connection corresponding to the stream

identifier has received more than configured percentage

(max_threshold) of bad packets, configured action is

triggered.

```
add stream identifier packetcreditrateidentifier packetcreditrateselector
-interval 1



 add responder policy packetcreditratesessionpolicy
"ANALYTICS.STREAM(\"packetcreditrateidentifier\").COLLECT_STATS(\"PACKET_C
REDITS\", 75, RESET)" NOOP
```

Valid mode: PACKET_CREDITS

Valid action: RESET/DROP

Valid max_threshold: 0-100

Parameters(expressions not allowed):

mode -

```
stream mode
```

max_threshold -

```
maximum threshold percentage of bad packets.
```

action -

```
Action
```

Returns: bool_at

COLLECT_STATS(text_tmode, uint max_threshold, stream_action_eaction, uint type)

Returns true if statistics collection on an object was

successful.

This expression is used to rate limit packets per second.

When an object/connection corresponding to the stream

identifier has received more than configured (max_threshold)

number of packets, configured action is triggered.

```
add stream identifier packetrateidentifier packetrateselector -interval 1



 add responder policy packetratesessionpolicy
"ANALYTICS.STREAM(\"packetrateidentifier\").COLLECT_STATS(\"PACKET_LIMIT\"
, 5000, RESET, 1)" NOOP
```

Valid mode: PACKET_LIMIT

Valid action: RESET/DROP

Valid max_threshold: strictly greater than zero.

Valid type: 0/1 0 for bursty 1 for smooth

Parameters(expressions not allowed):

mode -

```
stream mode
```

max_threshold -

```
maximum threshold of packets per second.
```

action -

```
Action
```

type -

```
rate limiting type like smooth/bursty.
```

Returns: bool_at

CONNECTIONS

Returns the number of connections on an object.

Returns: unsigned_long_at

IS_TOP_FREQUENTS(uint n)

Returns true if a given object is in the top n of a identifier.

Parameters(expressions not allowed):

n -

```
Given object must be in the top n frequents of the collected objects.
```

Returns: bool_at

IS_TOP(uint n)

Returns true if a given object is in the top n of a identifier.

Parameters(expressions not allowed):

n -

```
Given object must be in the top n of the collected objects.
```

Returns: bool_at

REQUESTS

Returns the number of requests on an object.

Returns: unsigned_long_at

RESPTIME

Returns the average response time of an object.

Returns: unsigned_long_at

STREAM_TRANSACTION_TRACK_MODE_E

This object provides stream analytics transaction tracking modes.

This topic lists the expressions that are provided by this class.

Enumerations

RESPONSE_TIME

SUBSCRIBER_T

Retrieves AVPs and other subscriber attributes from a subscriber session.

This topic lists the expressions that are provided by this class.

AVP(uint avpcode)

Returns the AVP with the given AVP code from the subscriber session.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

AVP(uint avpcode, uint vendorid)

Returns the AVP with the given code and within given Vendor ID namespace from this subscriber session.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor ID
```

Returns: avp_t

RULE_ACTIVE(text_trulename)

Returns true if given rulename is active for this subscriber session.

- Expression RULE_ACTIVE searches the current policies of subscriber and returns TRUE if it is present, otherwise returns FALSE.

Parameters(expressions not allowed):

rulename -

```
Polname
```

Returns: bool_at

SERVICEPATH

Returns the SERVICEPATH name as a string.

- Expression SERVICEPATH fetches the value in servicepathAVP for dynamic subscribers, or configured servicepath for static subscribers.
- It returns NULL if servicepathAVP is not present or servicepath is not configured.

Returns: servicepath_t

SYS_T

Designates an expression that refers to the ApplicationSwitch,

rather than the client or server.

This topic lists the expressions that are provided by this class.

CHECK_LIMIT(text_tidentifier_name)

Used for rate limiting.

Returns a boolean Result.

Parameters(expressions not allowed):

identifier_name -

```
Identifier Name
```

Returns: bool_at

CI_CALLOUT(ci_callout_bool cic)

Make a Content Inspection callout.

Returns a boolean result.

Parameters(expressions not allowed):

cic -

```
Name of Content Inspection callout that returns boolean result.
```

Returns: bool_at

CI_CALLOUT(ci_callout_num cic)

Make a Content Inspection callout.

Returns a number result.

Parameters(expressions not allowed):

cic -

```
Name of Content Inspection callout that returns number result.
```

Returns: num_at

CI_CALLOUT(ci_callout_text cic)

Make a Content Inspection callout.

Returns a text result.

Parameters(expressions not allowed):

cic -

```
Name of Content Inspection callout that returns text result.
```

Returns: text_t

CLIENT_EXPR(text_ts)

Evaluates a client expression.

Returns a boolean result.

Examples:

CLIENT_EXPR("client.av(Norton).exists")

add some more examples

Parameters(expressions not allowed):

s -

```
Client Expression
```

Returns: bool_at

ERROR

Returns the error encountered when performing the last

operation.

Returns: error_t

EVAL_CLASSIC_EXPR(classic_expr s)

Evaluate classic-style Citrix ADC expressions.

NOTE: use of Classic expressions is deprecated - use Default

(Advanced) expressions instead of this function.

The result of this evaluation can continued to be used

for further Advanced operations. This operation is intended to

be a bridge between traditional Citrix ADC expressions, which

we call "classic" expressions and the Advanced expressions. Both

inline and named classic expressions can be used as input.

Examples:

SYS.EVAL_CLASSIC_EXPR("ns_slowclient")

SYS.EVAL_CLASSIC_EXPR("REQ.SSL.CLIENT.CIPHER.BITS > 1000")

SYS.EVAL_CLASSIC_EXPR("url contains abc")

SYS.EVAL_CLASSIC_EXPR("REQ.IP.SOURCEIP == 10.102.1.61 -netmask 255.255.255.255")

SYS.EVAL_CLASSIC_EXPR("time >= *:30:00GMT")

SYS.EVAL_CLASSIC_EXPR("e1 || e2")

SYS.EVAL_CLASSIC_EXPR("REQ.HTTP.URLLEN > 50")

SYS.EVAL_CLASSIC_EXPR("dayofweek == wedGMT")

Parameters(expressions not allowed):

s -

```
Classic Expression
```

Returns: bool_at

HTTP_CALLOUT(http_callout_bool hc)

Make a http callout.

Returns a boolean result.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns boolean result.
```

Returns: bool_at

HTTP_CALLOUT(http_callout_num hc)

Make a http callout.

Returns a number result.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns number result.
```

Returns: num_at

HTTP_CALLOUT(http_callout_text hc)

Make a http callout.

Returns a text result.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns text as result.
```

Returns: text_t

NANO_TIME

Returns the current system time as the number of nanoseconds since "GMT Jan 1 1970 0h 0m 0.000000000s".

Accuracy may be at larger granularity than nanosecond resolution.

Returns: unsigned_long_at

NON_BLOCKING_HTTP_CALLOUT(http_callout_bool hc)

Make a non blocking http callout.

Returns a boolean result 'true'.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns boolean result that is ignored.
```

Returns: bool_at

NON_BLOCKING_HTTP_CALLOUT(http_callout_num hc)

Make a non blocking http callout.

Returns a number that is '0'.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns number which is ignored.
```

Returns: num_at

NON_BLOCKING_HTTP_CALLOUT(http_callout_text hc)

Make a non blocking http callout.

Returns a text that is of zero length.

Parameters(expressions not allowed):

hc -

```
Name of Http callout that returns text as result which is ignored.
```

Returns: text_t

NSIP

Returns the netscaler's IPV4 address.

Returns: ip_address_at

RANDOM

Returns a random number between 0 and 1,

inclusive of 0 but exclusive of 1

Returns: double_at

SERVICEGROUP(text_tservicegroup_name)

Used for Servicegroup expressions.These expression raises an undef if servicegroup with the given name doesn't exist.

Parameters(expressions not allowed):

servicegroup_name -

```
ServiceGroup Name
```

Returns: servicegroup_t

SERVICE(text_tservice_name)

Used for Service expressions.These expression raises an undef if service with the given name doesn't exist.

Parameters(expressions not allowed):

service_name -

```
Service Name
```

Returns: service_t

TIME

Returns the current system time as the number of seconds since "GMT Jan 1 1970 0h 0m 0s".

NOTE: It is not possible to set the system time to date and time before 1970.

Returns: time_at

UUID

Returns a newly generated UUID.

Returns: text_t

VSERVER(text_tvserver_name)

Used for vserver expressions.These expressions raise undef if virtual server with name doesn't exist.

Parameters(expressions not allowed):

vserver_name -

```
Virtual Server
```

Returns: vserver_t

TCP_OPTIONS_T

TCP Protocol Options

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

COUNT

Returns the value of number of tcp options present.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"
```

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

```
    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t
*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
tcp_options_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
```

```
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```


String or unsigned long value .)

Returns: unum_text_t

*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

TYPE_NAME(tcp_options_type_name_em)

Returns the value of tcp option as sequence of bytes
whose type is specified as an argument.

Parameters(expressions not allowed):

m -

```
TCP Option Type Name</p></div><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>TYPE</b>(uint type)</p><div><p>Returns the value of
tcp option as sequence of bytes
```

whose type is specified as an argument.

Parameters(expressions not allowed):

type -

```
Type value</p></div><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNQUOTE</b>(char c)</p><div><p>If selected
text is quoted by character c, Remove character c from
```

beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.
Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)
```

```
  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"
```

```
      in the first 1000 bytes of the body which is


        creator:{


           person:{


             name:'Milton'


           }


        }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


      This will return true as there is a node named


      'creator' with parent 'Book' in the first 1000 bytes of the


      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost',

      "price":"36"

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is
```

```
      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)


      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

TCP_OPTIONS_TYPE_NAME_E

This object represents one of the TCP Options Type Name.

This topic lists the expressions that are provided by this class.

Enumerations

MAXSEG

REPEATER

SACK_PERMITTED

TIMESTAMP

WINDOW

TCP_OPTIONS_TYPE_T

TCP Protocol Options Type

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns TRUE if an instance of the current tcp option type

exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
OPTION_LENGTH

Returns length of the current tcp option type in bytes

This is the total length of the TCP option that

got inserted in the TCP header and is not same

as the string length of the option value

Returns: num_at
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                      Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
```

```
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

(http_cache_control_t : This object provides for the ability to work on HTTP Cache-Control headers.) Returns: http_cache_control_t * TYPECAST_HTTP_CONTENT_LENGTH_T Convert text_t to http_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or HTTP.RES.CONTENT_LENGTH object can be

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
tcp_options_type_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton"</
p> Parameters(expressions not allowed): xpathex -

XPath Expression Returns: <a
href="text_t.html">text_t </body></html>

TCP_REPEATER_OPTION_T

TCP Protocol Option : Repeater

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

EXISTS

Returns TRUE if Repeater option type exists.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"
```

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

```
    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

IP

Returns IPV4 address present in the Repeater option.

Returns: ip_address_at

*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at

*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>MAC</b></p><div><p>Returns MAC address
present in the Repeater option.</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the value
corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
OPTION_LENGTH

Returns length of the Repeater option in bytes

This is the total length of the TCP option that

got inserted in the TCP header and is not same as

the string length of the option value

Returns: num_at
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +
```

```
"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB

- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t
*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
tcp_repeater_option_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


        The original JSON document is not disturbed but the


        part of the JSON body selected by the


        expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                          XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

TCP_REQ_PROTOCOL_T

TCP Protocol Request

This topic lists the expressions that are provided by this class.

BANDWIDTH

Returns the bandwidth of the TCP connection.

Returns: num_at

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

DSTPORT

Returns the ID of the current packet's destination port as a number.

Returns: num_at

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

MSS

Returns the maximum segment size (MSS) for the current connection

as a number.

Returns: num_at

OPTIONS

Returns the tcp options.

Returns: tcp_options_t

PAYLOAD(uint i)

Returns the specified portion of the TCP payload data as a string,

beginning with the first character and continuing for the number of

characters designated in uint.

Parameters:

i -

```
Body length
```

Returns: text_t

REPEATER_OPTION

Returns the branch repeater tcp option.

Returns: tcp_repeater_option_t

SMOOTHRTT

Returns the smoothed round trip time of the TCP connection.

Returns: num_at

SRCPORT

Returns the ID of the current packet's source port as a number.

Returns: num_at

TCP_RES_PROTOCOL_T

TCP Protocol Response

This topic lists the expressions that are provided by this class.

BANDWIDTH

Returns the bandwidth of the TCP connection.

Returns: num_at

DSTPORT

Returns the ID of the current packet's destination port as a number.

Returns: num_at

MSS

Returns the maximum segment size (MSS) for the current connection

as a number.

Note: The MSS on the server side connection is derived using

internal Citrix ADC logic. Typically users cannot attribute

any meaning to the MSS value on the server connection and

should not use this expression in policies.

Returns: num_at

OPTIONS

Returns the tcp options.

Returns: tcp_options_t

PAYLOAD(uint i)

Returns the specified portion of the TCP payload data as a string,

beginning with the first character and continuing for the number of

characters designated in uint.

Parameters:

i -

```
Body length
```

Returns: text_t

SMOOTHRTT

Returns the smoothed round trip time of the TCP connection.

Returns: num_at

SRCPORT

Returns the ID of the current packet's source port as a number.

Returns: num_at

TCP_STATE_E

TCP states

This topic lists the expressions that are provided by this class.

Enumerations

CLOSED

CLOSE_WAIT

CLOSING

ESTABLISHED

FIN_WAIT_1

FIN_WAIT_2

LAST_ACK

LISTEN

NOT_APPLICABLE

SYN_RECEIVED

SYN_SENT

TIME_WAIT

Expressions

EQ(tcp_state_estate)

Returns Boolean True if the TCP state matches the argument.

Example: CONNECTION.TCPSTATE.EQ(ESTABLISHED)

Parameters(expressions not allowed):

state -

```
TCP State
```

Returns: bool_at

NE(tcp_state_estate)

Returns Boolean True if the TCP state do not matches the argument.

Example: CONNECTION.TCPSTATE.NE(ESTABLISHED)

Parameters(expressions not allowed):

state -

```
TCP State
```

Returns: bool_at

TEXTMODE_E

Text Mode Enum

This topic lists the expressions that are provided by this class.

Enumerations

BACKSLASH_ENCODED

BAD_ENCODE_RAISE_UNDEF

IGNORECASE

NOIGNORECASE

NOURLENCODED

NO_BACKSLASH_ENCODED

NO_BAD_ENCODE_RAISE_UNDEF

NO_PLUS_AS_SPACE

PLUS_AS_SPACE

URLENCODED

TEXT_T

Designates any piece of text.

This topic lists the expressions that are provided by this class.

AFTER_REGEX(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

AFTER_STR_ANY(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

AFTER_STR(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

APPEND(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

APPEND(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

APPEND(ip_address_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

APPEND(num_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

APPEND(time_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

APPEND(unsigned_long_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

APPEND(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

APPEND(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

APPEND(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

APPEND(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

APPEND(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

APPEND(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

APPEND(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

APPEND(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

APPEND(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

B64DECODE

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

B64ENCODE

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

BEFORE_REGEX(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

BEFORE_STR_ANY(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

BEFORE_STR(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

BETWEEN(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

BIN_DECODE(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

BIN_ENCODE(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

BLOB_TO_HEX

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

COMPARE(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

CONTAINS_ANY(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

CONTAINS_INDEX(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

CONTAINS(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

CRC32

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

DECODE_USING_TEXT_MODE

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

DECRYPT

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

DECRYPT(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

DECRYPT(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

DECRYPT(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

DIGEST(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

ENCRYPT

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

ENCRYPT(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

ENCRYPT(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

ENCRYPT(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

ENDSWITH_ANY(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

ENDSWITH_INDEX(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

ENDSWITH(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

EQUALS_ANY(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

EQUALS_INDEX(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

EQ(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

GET_SIGNED16(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

GET_SIGNED32(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

GET_SIGNED8(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

GET_UNSIGNED16(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

GET_UNSIGNED32(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

GET_UNSIGNED64(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

GET_UNSIGNED8(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

GET_URLSET_METADATA(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>GT</b>(<a href="text_t.html">text_t</
a>s)</p><div><p>Results in Boolean TRUE value if the Text object is
greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>HASH</b></p><div><p>Returns a positive
31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at

HMAC(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
HMAC(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform
and select text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

HTTP_HEADER_SAFE

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

HTTP_URL_SAFE

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"
```

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

```
    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
IS_BOOL_AT

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
IS_DOUBLE_AT

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
IS_IPV6_AT

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
IS_IPV6_SUBNET_AT

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

IS_IP_AT

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

IS_MAC_ADDRESS_AT

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

IS_NUM_AT

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

IS_NUM_AT(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".

- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if
the Text follows any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at

IS_UNSIGNED_LONG_AT

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at

IS_UNSIGNED_LONG_AT(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
JWT_VERIFY_XML(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

LENGTH

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

LE(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>LOCATION_LONG</b></p><div><p>Get the
location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
LT(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the value
corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>MAP_STRING</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns value corresponding to key in the
string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe
without applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>MAX</
b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the string
argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>MIN</b>(<a href="text_t.html">text_t</
a>s)</p><div><p>Compares the string argument with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>NE</b>(<a href="text_t.html">text_t</
```

```
a>s)</p><div><p>Results in Boolean TRUE value if the Text object is not
equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>NORMALIZE_URL</b></p><div><p>Normalize
URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
PKEY_ENCRYPT_PEM_NO_PADDING(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

PKEY_ENCRYPT_PEM(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

PKEY_SIGN_CERTKEY(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
PKEY_SIGN_XML(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
PKEY_VERIFY_CERTKEY(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.
Returns: bool_at
PKEY_VERIFY_XML(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
PREFIX(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the
Text. The length of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>REGEX_MATCH</b>(re regex)</
p><div><p>Returns TRUE if and only if the text contains at least one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text
that matches the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>REPEAT</b>(uint i)</p><div><p>Produces a string which
contains 'i' sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SET_CHAR_SET</b>(<a href="charset_e.html">charset_e</
a>charset)</p><div><p>Set the character set to use for subsequent
functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```


many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>SKIP</
b>(uint i)</p><div><p>Select the Text after skipping over a prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</
a>patset_name)</p><div><p>Returns the numerical index of the matching
string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></
div><div><div><p><b>STARTSWITH</b>(<a href="text_t.html">text_t</
a>start)</p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>STRIP_END_WS</b></p><div><p>Selects the
Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

STRIP_START_CHARS(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>STRIP_START_WS</b></p><div><p>Selects the
Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

SUBSTR_ANY(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the
Text object of the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Select the first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the
Text. The length of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>TO_LOWER</b></p><div><p>Converts
the selected text to lower case.<br /><br /><br /><br />For example, the
string "ABCd" is converted to "abcd"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>TO_UPPER</b></p><div><p>Converts the selected text to
upper case.<br /><br /><br /><br />For example, the string "abcD" is
converted to "ABCD"</p><p><b>Returns: </b><a href="text_t.html">text_t</
```

```
a></p></div></div></div><div><div><p><b>TRUNCATE</b>(char c, uint i)</
p><div><p>Select the Text after truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after
truncating a suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

TYPECAST_AVP_UNUM_T

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
TYPECAST_BOOL_AT

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at
TYPECAST_COOKIE_T

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

TYPECAST_DIAMETER_FLAG_T

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

TYPECAST_DNS_DOMAINNAME_T

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t

TYPECAST_DNS_FLAG_T

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t

TYPECAST_DOUBLE_AT

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at

TYPECAST_DOUBLE_T(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
```

```
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></
div><div><div><p><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></
div><div><div><p><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></
div><div><div><p><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></
div><div><div><p><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
```

```
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></
div><div><div><p><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert
text_t to http_cache_control_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

TYPECAST_HTTP_COOKIE_T

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

TYPECAST_HTTP_DATE_T

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

TYPECAST_HTTP_HEADER_T(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>TYPECAST_HTTP_HOSTNAME_T</b></p><div><p>Convert text_t
to http_hostname_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_hostname_t : This object provides operations for parsing the HTTP
hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

TYPECAST_HTTP_METHOD_T

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

TYPECAST_HTTP_SET_COOKIE_T

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

TYPECAST_HTTP_URL_T

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

TYPECAST_HTTP_VERSION_T

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

TYPECAST_IPV6_ADDRESS_AT

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

using the '+' operator. The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions. No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression. Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB. When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean. Further, when

A and/or B is an IPv6 address, the only operator allowed between

them is '+'.

```
<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>TYPECAST_IPV6_ADDRESS_T</b></p><div><p>Convert text_t
to ipv6_address_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (ipv6_address_t :
This object represents an IPv6 address present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

TYPECAST_IPV6_SUBNET_AT

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at

TYPECAST_IPV6_SUBNET_T

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
TYPECAST_IP_ADDRESS_AT

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
TYPECAST_IP_ADDRESS_T

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
TYPECAST_LIST_T(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert text_t
to mac_address_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (mac_address_t :
Represents a MAC address present in the 12:34:56:78:9a:bc
```


format.)

Returns: mac_address_t

TYPECAST_NUM_AT

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

TYPECAST_NUM_T(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></
div><div><div><p><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></
div><div><div><p><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char
quote)</p><div><p>Convert text_t to nvlist_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (nvlist_t : Two character delimiters and quote-character
have to be provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></
div><div><div><p><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

TYPECAST_RADIUS_AVP_VALUE_T

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t

TYPECAST_SEQ_TEXT_T

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t

TYPECAST_SERVICEPATH_T

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

TYPECAST_SIP_CALLID_HEADER_T

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t

TYPECAST_SIP_CONTENT_LENGTH_T

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

TYPECAST_SIP_FROM_HEADER_T

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t
TYPECAST_SIP_HEADER_T(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>TYPECAST_SIP_HOSTNAME_T</b></p><div><p>Convert text_t
to sip_hostname_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (sip_hostname_t :
This object provides operations for parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t
TYPECAST_SIP_METHOD_T

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
TYPECAST_SIP_TO_HEADER_T

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

TYPECAST_SIP_URL_T

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

TYPECAST_SIP_VERSION_T

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

TYPECAST_SIP_VIA_HEADER_T

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

TYPECAST_SSLVPN_MODE_T

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

TYPECAST_SSL_TEXT_T(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>TYPECAST_TCP_OPTIONS_T</b></
p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></p><div><p>Convert
text_t to tcp_options_type_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(tcp_options_type_t : TCP Protocol Options Type)</p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>TYPECAST_TCP_REPEATER_OPTION_T</b></p><div><p>Convert
text_t to tcp_repeater_option_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(tcp_repeater_option_t : TCP Protocol Option : Repeater)</p><p><b>Returns:
</b><a href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></
div></div></div><div><div><p><b>TYPECAST_TIME_AT</b></p><div><p>Convert
text_t to time_at. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (time_at : Treats a
number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at

TYPECAST_TIME_T

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
TYPECAST_UNSIGNED_LONG_AT

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
TYPECAST_UNSIGNED_LONG_T(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
```

```
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert text_t to
unum_text_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (unum_text_t : Provides
operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t

TYPECAST_URL_CATEGORY_T

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t

TYPECAST_VENDOR_SPEC_APPID_AVP_T

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests: DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses: DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

TYPECAST_VENDOR_SPEC_RADIUS_AVP_T

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

UNQUOTE(char c)

If selected text is quoted by character c, Remove character c from beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

```
Dot: "."

Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
XML_DECRYPT(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
XML_ENCRYPT(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

XML_ENCRYPT(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

XPATH_HTML_WITH_MARKUP(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies
an XPath expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>XPATH_HTML</b>(xpath_double xpathex)</
p><div><p>Applies an XPath expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies
an XPath expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json_with_markup
operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }

    The original JSON document is not disturbed but the
```

```
      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>XPATH_JSON</b>(xpath_bool xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'
```

```
            }


        },


        "title":'Paradise Lost'


    }


}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)


    This will return true as there is a node named


    'creator' with parent 'Book' in the first 1000 bytes of the


    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>XPATH_JSON</b>(xpath_double xpathex)</
```

```
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>XPATH</
b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression evaluation. xpath
operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>XPATH</b>(xpath_double xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>XPATH</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

THIS_T

This is a special marker class.If the return type of a method is this_t then the current context's type does not change.

This topic lists the expressions that are provided by this class.

TIME_AT

Treats a number value as number of seconds since GMT Jan 1

1970 0h 0m 0s.

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from num_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

time value.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value is between

the Lower and the Upper time arguments. Both the Lower and

the Upper time arguments must be fully specified.

For example -

GMT 1995 Jan is fully specified. But GMT Jan or GMT 1995 20

or GMT Jan Mon_2 are not fully specified. The Upper and

Lower arguments must be both GMT or both Local. The day of

the week must not be present in either the Lower or the

Upper argument. It is OK to specify the day of the month as

the 1st, 2nd, 3rd or 4th weekday of the month (example Wed_3

is the 3rd Wednesday of the month).

The Upper Time argument must be bigger than the Lower Time

argument.

Assume the current time value is GMT 2005 May 1

10h 15m 30s. This is the 1st Sunday of the month of May in 2005.

BETWEEN(GMT 2004, GMT 2006): TRUE BETWEEN(GMT 2004 Jan, GMT 2006 Nov): TRUE BETWEEN(GMT 2004 Jan, GMT 2006): TRUE BETWEEN(GMT 2005 May Sun_1, GMT 2005 May Sun_3): TRUE BETWEEN(GMT 2005 May 1, GMT May 2005 1): TRUE BETWEEN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
Lower Time
```

time2 -

```
Upper Time
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_attime1, time_attime2)

Returns Boolean TRUE if the time value is greater than or

equal to the Lower argument and the time value is lesser

than or equal to the Upper argument.

Parameters:

time1 -

```
Lower GMT Time Value
```

time2 -

```
Upper GMT Time Value
```

Returns: bool_at

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DAY

Extracts the day of the month from the current system time, and

returns that value as a number from 1 through 31 representing that

day.

Returns: num_at

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

EQ(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

EQ(GMT 2005): TRUE EQ(GMT 2005 Dec): FALSE EQ(Local 2005 May): TRUE or FALSE depending on the current

timezone. EQ(GMT 10h): TRUE EQ(GMT 10h 30s): TRUE EQ(GMT May 10h): TRUE EQ(GMT Sun): TRUE EQ(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

EQ(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** GE**(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GE**(num_ati)

Returns Boolean TRUE if the number value is greater than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

GE(time_att)

Results in Boolean TRUE value if the time value is greater

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

GE(GMT 2004): TRUE GE(GMT 2005 Jan): TRUE GE(Local 2005 May): TRUE or FALSE depending on the current

timezone. GE(GMT 8h): TRUE GE(GMT 30m): FALSE GE(GMT May 10h): TRUE GE(GMT May 10h 0m): TRUE GE(GMT Sun): TRUE GE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

GE(time_att)

Results in Boolean TRUE value if the time value is

greater than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** GT**(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** GT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GT**(num_ati)

Returns Boolean TRUE if the number value is greater than the

argument.

Parameters:

i -

```
Number
```

Returns: bool_at

GT(time_att)

Results in Boolean TRUE value if the time value is greater

than the argument time.

Assume the current time value is GMT

2005 May 1 10h 15m 30s. This is the 1st Sunday of the month

of May in 2005.

GT(GMT 2004): TRUE GT(GMT 2005 Jan): TRUE GT(Local 2005 May): TRUE or FALSE depending on the current

timezone. GT(GMT 8h): TRUE GT(GMT 30m): FALSE GT(GMT May 10h): FALSE GT(GMT May 10h 0m): TRUE GT(GMT Sun): FALSE GT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

GT(time_att)

Results in Boolean TRUE value if the time value is

greater than the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

HOURS

Extracts the hour from the current system time, and returns that

value as an integer from 0 to 23.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LE**(num_ati)

Returns Boolean TRUE if the number value is lesser than or

equal to the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

LE(time_att)

Results in Boolean TRUE value if the time value is lesser

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LE(GMT 2006): TRUE LE(GMT 2005 Dec): TRUE LE(Local 2005 May): TRUE or FALSE depending on the current

timezone. LE(GMT 8h): FALSE LE(GMT 30m): TRUE LE(GMT May 10h): TRUE LE(GMT Jun 11h): TRUE LE(GMT Wed): TRUE LE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

LE(time_att)

Results in Boolean TRUE value if the time value is

less than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length
```

Returns: num_at

*** LT**(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** LT**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** LT**(num_ati)

Returns Boolean TRUE if the number value is lesser than

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

LT(time_att)

Results in Boolean TRUE value if the time value is lesser

than the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LT(GMT 2006): TRUE LT(GMT 2005 Dec): TRUE LT(Local 2005 May): TRUE or FALSE depending on the current

timezone. LT(GMT 8h): FALSE LT(GMT 30m): TRUE LT(GMT May 10h): FALSE LT(GMT Jun 11h): TRUE LT(GMT Wed): TRUE LT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

LT(time_att)

Results in Boolean TRUE value if the time value is

less than the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** MAX**(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MAX**(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MAX**(num_ati)

Return the maximum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MAX**(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MINUTES

Extracts the minute from the current system time, and returns that

value as an integer from 0 to 59.

Returns: num_at

*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MIN**(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MIN**(num_ati)

Return the minimum value after comparing the argument

with the current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MIN**(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MOD**(num_ati)

Return the Number Value that is the renaminder after dividing the

current number value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** MOD**(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MONTH

Extracts the month from the current system time, and returns that

value as an integer from 1 (January) to 12 (December).

Returns: num_at

*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** MUL**(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** MUL**(num_ati)

Return the Number Value after multiplying the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** MUL**(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Number Value after negating the current number

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** NE**(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** NE**(num_ati)

Returns Boolean TRUE if the number value is not equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

NE(time_att)

Results in Boolean TRUE value if the time value is

not equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

NE(GMT 2005): FALSE NE(GMT 2005 Dec): TRUE NE(Local 2005 May): TRUE or FALSE depending on the current

timezone. NE(GMT 10h): FALSE NE(GMT 10h 30s): FALSE NE(GMT May 10h): FALSE NE(GMT Sun): FALSE NE(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

NE(time_att)

Results in Boolean TRUE value if the time value is

not equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

RELATIVE_BOOT

Calculates the number of seconds to the closest previous or scheduled

reboot, and returns an integer. If the closest boot time is in the past,

the integer is negative; if it is in the future, the integer is positive.

Returns: num_at

RELATIVE_NOW

Calculates the number of seconds between the current system time and

the specified time, and returns an integer showing that number. If the

designated time is in the past, the integer is negative; if it is in

the future, the integer is positive.

Returns: num_at

*** RSHIFT**(uint i)

Return the Number Value after bitwise right shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32. For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length
```

Returns: num_at

SECONDS

Extracts the seconds from the current system time, and returns that

value as an integer from 0 to 59.

Returns: num_at

*** SIGNED16_STRING**(endian_eendianness)

Create a signed 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED32_STRING**(endian_eendianness)

Create a signed 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** SIGNED8_STRING**

Create a signed 8-bit binary string from the number.

Returns: text_t

*** SUB**(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** SUB**(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** SUB**(num_ati)

Return the Number Value after subtracting the argument from the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** SUB**(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at

*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at

TYPECAST_NUM_AT

Convert time_at to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at

TYPECAST_TEXT_T

Typecast the Time Value to its string representation.

Returns: text_t

*** TYPECAST_TIME_AT**

Typecast the Number Value to a Time value.

Returns: time_at

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at

*** UNSIGNED16_STRING**(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED32_STRING**(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED64_STRING**(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

*** UNSIGNED8_STRING**

Create an unsigned 8-bit binary string from the number.

Returns: text_t

WEEKDAY

Results in a Number Value between 0 (Sunday) and 6

(Saturday) which gives the week day in the time value.

Returns: num_at

*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t

*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

WITHIN(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value lies

within all the ranges defined by the Lower and Upper time

arguments.

If an element of time like day or hour is left

unspecified in the Lower argument then it is assumed to have the

lowest value of its range. If an element is left

unspecified in the Upper argument then it is assumed to have the

highest value of its range. If the year is specified in one

of the argument then it must be specified in the other argument as

well. The ranges for different elements of time are month

1-12, day 1-31, weekday 0-6, hour 0-23, minutes 0-59 and

seconds 0-59.

Each element of time defines a range. For the result to be

TRUE each element of time in the time value must lie in the

corresponding range present in the Lower and Upper

arguments.

Assume the time value is GMT 2005 May 10 10h 15m 30s. This

is the second Tuesday of the month. WITHIN(GMT 2004, GMT 2006): TRUE WITHIN(GMT 2004 Jan, GMT 2006 Mar): FALSE (May doesn't fall in

the Jan-Mar range.) WITHIN(GMT Feb, GMT): TRUE (May falls in the Feb-Dec range.) WITHIN(GMT Sun_1, GMT Sun_3): TRUE (2nd Tuesday lies within

1st Sunday and the 3rd Sunday.) WITHIN(GMT 2005 May 1 10h, GMT May 2005 1 17h): TRUE WITHIN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
Lower Time
```

time2 -

```
Upper Time
```

Returns: bool_at

YEAR

Extracts the year from the current system time, and returns that

value as a four-digit integer.

Returns: num_at

TIME_T

This object treats a Text object as a date string. The formats

supported are

1. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT
2. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT
3. ASCTIME :: Sun Nov 6 08:49:37 1994
4. ASCTIME like :: Sun Nov 6 08:49:37 1994
5. ASCTIME like :: Sun Nov 06 08:49:37 1994
6. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from text_t, time_at.

*** ABS**

Return the absolute value of the number.

Returns: num_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Number Value after adding the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

time value.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the number value is greater than or

equal to the Lower argument and the number value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value is between

the Lower and the Upper time arguments. Both the Lower and

the Upper time arguments must be fully specified.

For example -

GMT 1995 Jan is fully specified. But GMT Jan or GMT 1995 20

or GMT Jan Mon_2 are not fully specified. The Upper and

Lower arguments must be both GMT or both Local. The day of

the week must not be present in either the Lower or the

Upper argument. It is OK to specify the day of the month as

the 1st, 2nd, 3rd or 4th weekday of the month (example Wed_3

is the 3rd Wednesday of the month).

The Upper Time argument must be bigger than the Lower Time

argument.

Assume the current time value is GMT 2005 May 1

10h 15m 30s. This is the 1st Sunday of the month of May in 2005.

BETWEEN(GMT 2004, GMT 2006): TRUE BETWEEN(GMT 2004 Jan, GMT 2006 Nov): TRUE BETWEEN(GMT 2004 Jan, GMT 2006): TRUE BETWEEN(GMT 2005 May Sun_1, GMT 2005 May Sun_3): TRUE BETWEEN(GMT 2005 May 1, GMT May 2005 1): TRUE BETWEEN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
Lower Time
```

time2 -

```
Upper Time
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_attime1, time_attime2)

Returns Boolean TRUE if the time value is greater than or

equal to the Lower argument and the time value is lesser

than or equal to the Upper argument.

Parameters:

time1 -

```
Lower GMT Time Value
```

time2 -

```
Upper GMT Time Value
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Number Value after bitwise ANDing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Number Value after bitwise negating the current number

value.

Returns: num_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Number Value after bitwise ORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Number Value after bitwise XORing the argument to the

current number value.

Parameters:

i -

```
Number
```

Returns: num_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DAY**

Extracts the day of the month from the current system time, and

returns that value as a number from 1 through 31 representing that

day.

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Number Value after dividing the current number

value by the argument.

Parameters:

i -

```
Number
```

Returns: num_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the number value is equal to

the argument.

Parameters:

i -

```
Number
```

Returns: bool_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

EQ(GMT 2005): TRUE EQ(GMT 2005 Dec): FALSE EQ(Local 2005 May): TRUE or FALSE depending on the current

timezone. EQ(GMT 10h): TRUE EQ(GMT 10h 30s): TRUE EQ(GMT May 10h): TRUE EQ(GMT Sun): TRUE EQ(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time
```

Returns: bool_at

*** EQ**(time_att)

Results in Boolean TRUE value if the time value is

equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
```

```
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is greater
```

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

GE(GMT 2004): TRUE

GE(GMT 2005 Jan): TRUE

GE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

GE(GMT 8h): TRUE

GE(GMT 30m): FALSE

GE(GMT May 10h): TRUE

GE(GMT May 10h 0m): TRUE

GE(GMT Sun): TRUE

GE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

greater than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>GT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is greater than the
```

argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is greater
```

than the argument time.

Assume the current time value is GMT

2005 May 1 10h 15m 30s. This is the 1st Sunday of the month

of May in 2005.

GT(GMT 2004): TRUE

GT(GMT 2005 Jan): TRUE

GT(Local 2005 May): TRUE or FALSE depending on the current

timezone.

GT(GMT 8h): TRUE

GT(GMT 30m): FALSE

GT(GMT May 10h): FALSE

GT(GMT May 10h 0m): TRUE

GT(GMT Sun): FALSE

GT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

greater than the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>HASH</b></p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HOURS</b></p><div><p>Extracts the hour from the current system
time, and returns that
```

value as an integer from 0 to 23.

Returns: num_at

*** HTML_XML_SAFE**

Apply XML safe transform and select text.

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"
```

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

```
    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at
*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at
*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is lesser
```

than or equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LE(GMT 2006): TRUE

LE(GMT 2005 Dec): TRUE

LE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

LE(GMT 8h): FALSE

LE(GMT 30m): TRUE

LE(GMT May 10h): TRUE

LE(GMT Jun 11h): TRUE

LE(GMT Wed): TRUE

LE(GMT May Sun_1): TRUE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

less than or equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>LOCATION_LONG</b></p><div><p>Get the location qualifier from
CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LSHIFT**(uint i)

Return the Number Value after bitwise left shifting the current

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.LSHIFT(1) = HTTP.REQ.URL.LENGTH.LSHIFT(33) = 8,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>LT</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Returns
Boolean TRUE if the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is lesser than
```

the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is less than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is lesser
```

than the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

LT(GMT 2006): TRUE

LT(GMT 2005 Dec): TRUE

LT(Local 2005 May): TRUE or FALSE depending on the current

timezone.

LT(GMT 8h): FALSE

LT(GMT 30m): TRUE

LT(GMT May 10h): FALSE

LT(GMT Jun 11h): TRUE

LT(GMT Wed): TRUE

LT(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

less than the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
```

```
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="double_at.html">double_at</a>i)</p><div><p>Return
the maximum double value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the maximum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the maximum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MINUTES</b></p><div><p>Extracts the minute
from the current system time, and returns that
```

value as an integer from 0 to 59.

Returns: num_at
*** MIN**(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum value after comparing the argument
```

with the current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the minimum unsigned
long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
```

```
p><div><p>Return the Number Value that is the renaminder after dividing
the
```

current number value by the argument.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MOD</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MONTH</b></p><div><p>Extracts the month from
the current system time, and returns that
```

value as an integer from 1 (January) to 12 (December).

Returns: num_at
*** MUL**(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
```

```
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after multiplying the argument to the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Number Value
after negating the current number
```

value.

Returns: num_at

*** NE**(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
number value is not equal to
```


the argument.

Parameters:

i -


```
Number</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```


input argument.

Parameters:

s -


```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```


not equal to the argument time.

Assume the current time

value is GMT 2005 May 1 10h 15m 30s. This is the 1st Sunday

of the month of May in 2005.

NE(GMT 2005): FALSE

NE(GMT 2005 Dec): TRUE

NE(Local 2005 May): TRUE or FALSE depending on the current

timezone.

NE(GMT 10h): FALSE

NE(GMT 10h 30s): FALSE

NE(GMT May 10h): FALSE

NE(GMT Sun): FALSE

NE(GMT May Sun_1): FALSE

Parameters(expressions not allowed):

t -

```
Time</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>t)</p><div><p>Results in Boolean TRUE value
if the time value is
```

not equal to the argument GMT time value.

Parameters:

t -

```
GMT Time Value</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>NORMALIZE_URL</b></p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t

*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyValue> element with

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)

- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                      Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>RELATIVE_BOOT</b></p><div><p>Calculates the number of seconds to the
closest previous or scheduled
```

reboot, and returns an integer. If the closest boot time is in the past,

the integer is negative; if it is in the future, the integer is positive.

Returns: num_at
*** RELATIVE_NOW**

Calculates the number of seconds between the current system time and

the specified time, and returns an integer showing that number. If the

designated time is in the past, the integer is negative; if it is in

the future, the integer is positive.

Returns: num_at
*** REPEAT**(uint i)

Produces a string which contains 'i' sequential copies of the input

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Number Value after bitwise right shifting the current
```

number value by the argument number of bits. Note that the number

of bits shifted is i modulo 32.

For example,

let HTTP.REQ.URL.LENGTH be 4. Then,

HTTP.REQ.URL.LENGTH.RSHIFT(1) = HTTP.REQ.URL.LENGTH.RSHIFT(33) = 2,

since 33 on division by 32 yields a remainder of 1.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>SECONDS</b></p><div><p>Extracts the seconds from the current system
time, and returns that
```

value as an integer from 0 to 59.

Returns: num_at

*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a href="time_t.html">time_t</
a></p></div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

"http://"

2. %uWWHH will be translated to the character sequence YZ where WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.
   For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.
For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.
   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.
   For example, the encoded text "http\x3a//" will be decoded to

"http://"

3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
                        Text Mode</p></div><p><b>Returns: </b><a
href="time_t.html">time_t</a></p></div></div></div><div><div><p><b>* </
b><b>SIGNED16_STRING</b>(<a href="endian_e.html">endian_e</a>endianness)</
p><div><p>Create a signed 16-bit binary string from the number.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create a signed
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SIGNED8_STRING</b></p><div><p>Create a signed
8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(char c, uint i)</p><div><p>Select the Text after skipping
over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -
```

```
                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Number Value after subtracting the argument from the
```

current number value.

Parameters:

i -

```
Number</p></div><p><b>Returns: </b><a href="num_at.html">num_at</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Return the Unsigned Long
Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Number Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

(http_cache_control_t : This object provides for the ability to work on HTTP Cache-Control headers.) Returns: http_cache_control_t * TYPECAST_HTTP_CONTENT_LENGTH_T Convert text_t to http_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or HTTP.RES.CONTENT_LENGTH object can be

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Typecast the Number Value to an IP Address value.

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
TYPECAST_NUM_AT

Convert time_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert time_t to
text_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (text_t : Designates any piece
of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>TYPECAST_TIME_AT</b></p><div><p>Convert
time_t to time_at. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (time_at : Treats a
number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Number Value to a Unsigned Long value.

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```


supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -


```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```


supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
```

```
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

```
    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** WEEKDAY**

Results in a Number Value between 0 (Sunday) and 6

(Saturday) which gives the week day in the time value.

Returns: num_at
*** WEEKDAY_STRING**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the long form of the string name for that day, e.g.

0 = "Sunday", 6 = "Saturday".

Returns: text_t
*** WEEKDAY_STRING_SHORT**

Given the numeric day of the week, e.g. SYS.TIME.WEEKDAY,

return the short form of the string name for that day, e.g.

0 = "Sun", 6 = "Sat".

Returns: text_t

*** WITHIN**(time_attime1, time_attime2)

Results in a Boolean TRUE value if the time value lies

within all the ranges defined by the Lower and Upper time

arguments.

If an element of time like day or hour is left

unspecified in the Lower argument then it is assumed to have the

lowest value of its range.

If an element is left

unspecified in the Upper argument then it is assumed to have the

highest value of its range.

If the year is specified in one

of the argument then it must be specified in the other argument as

well.

The ranges for different elements of time are month

1-12, day 1-31, weekday 0-6, hour 0-23, minutes 0-59 and

seconds 0-59.

Each element of time defines a range. For the result to be

TRUE each element of time in the time value must lie in the

corresponding range present in the Lower and Upper

arguments.

Assume the time value is GMT 2005 May 10 10h 15m 30s. This

is the second Tuesday of the month.

WITHIN(GMT 2004, GMT 2006): TRUE

WITHIN(GMT 2004 Jan, GMT 2006 Mar): FALSE (May doesn't fall in

the Jan-Mar range.)

WITHIN(GMT Feb, GMT): TRUE (May falls in the Feb-Dec range.)

WITHIN(GMT Sun_1, GMT Sun_3): TRUE (2nd Tuesday lies within

1st Sunday and the 3rd Sunday.)

WITHIN(GMT 2005 May 1 10h, GMT May 2005 1 17h): TRUE

WITHIN(LOCAL 2005 May 1, LOCAL May 2005 1): Answer depends

on the Citrix ADC system's timezone.

Parameters(expressions not allowed):

time1 -

```
                        Lower Time</p><p>time2 -



                        Upper Time</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XML_DECRYPT</b>(<a href="text_t.html">text_t</a>certKeyName)</
p><div><p>Decrypt the selected content, which was encrypted using the W3C
XML
```

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



returns the &lt;title&gt; element in the document's



&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)
```

```
  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }
```

```
      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the

    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                    XPath Boolean Expression</p></div><p><b>Returns:
```

```
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

For the JSON file:

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)
```

```
      This expression when operated on the above XML file,


      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```


returns boolean value.

Examples:

```
  For the XML file:



  &lt;book&gt;&lt;creator&gt;&lt;person&gt;&lt;name&gt;Milton&lt;/name&gt;&lt;/
person&gt;&lt;/creator&gt;&lt;title&gt;Paradise Lost&lt;/title&gt;&lt;/
book&gt;HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book&gt;&lt;creator&gt;&lt;person&gt;&lt;name&gt;Milton&lt;/name&gt;&lt;/
person&gt;&lt;/creator&gt;&lt;title&gt;Paradise Lost&lt;/title&gt;&lt;price&gt;36&lt;/
price&gt;&lt;/book&gt;HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>YEAR</b></p><div><p>Extracts the year from the current system time,
and returns that
```


value as a four-digit integer.

Returns: num_at

</body></html>

TRAFFIC_DOMAIN_REQ_T

TRAFFIC_DOMAIN Request

This topic lists the expressions that are provided by this class.

ID

Returns Traffic Domain Identifier through which the current packet entered

the Application Switch as a number.

Returns: num_at

UDP_DNS_REQ_PROTOCOL_T

DNS Protocol

This topic lists the expressions that are provided by this class.

DOMAIN

Returns domain name.

Returns: dns_domainname_t

IS_AAAAREC

Returns TRUE if it is of AAAA record type.

Returns: bool_at

IS_ANYREC

Returns TRUE if it is of ANY record type.

Returns: bool_at

IS_AREC

Returns TRUE if it is of A record type.

Returns: bool_at

IS_CNAMEREC

Returns TRUE if it is of CNAME record type.

Returns: bool_at

IS_MXREC

Returns TRUE if it is of MX record type.

Returns: bool_at

IS_NAPTRREC

Returns TRUE if it is of NAPTR record type.

Returns: bool_at

IS_NSREC

Returns TRUE if it is of NS record type.

Returns: bool_at

IS_PTRREC

Returns TRUE if it is of PTR record type.

Returns: bool_at

IS_SOAREC

Returns TRUE if it is of SOA record type.

Returns: bool_at

IS_SRVREC

Returns TRUE if it is of SRV record type.

Returns: bool_at

UDP_REQ_PROTOCOL_T

UDP Protocol

This topic lists the expressions that are provided by this class.

CS_VSERVER

Returns the CS vserver serving the current request.

Returns: vserver_t

DNS

Returns DNS data for the current packet's UDP source port.

Returns: udp_dns_req_protocol_t

DSTPORT

Returns the ID of the current packet's UDP destination port as a number.

Returns: num_at

LB_VSERVER

Returns the LB vserver serving the current request.

Returns: vserver_t

RADIUS

Returns RADIUS data for the current packet

Returns: radius_protocol_t

SRCPORT

Returns the ID of the current packet's UDP source port as a number.

Returns: num_at

UNSIGNED_LONG_AT

Unsigned Long(64bit) Value

This topic lists the expressions that are provided by this class.

*** ABS**

Return absolute double value of the double value.

Returns: double_at

ADD(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

ADD(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

ADD(num_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

ADD(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

ADD(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

APPEND(text_tstr)

Appends the given string to the string representation of the current

unsigned long.

Parameters:

str -

```
the string to append.
```

Returns: text_t

AS_MICROSEC_STR

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

AS_MILLISEC_STR

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

AS_NANOSEC_STR

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

BETWEEN(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BETWEEN(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

BITAND(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITAND(num_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITAND(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITAND(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITNEG

Return the Unsigned Long Value after bitwise negating the current

unsigned long value.

Returns: unsigned_long_at

BITOR(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITOR(num_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITOR(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITOR(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITXOR(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITXOR(num_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITXOR(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

BITXOR(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DIV(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

DIV(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DIV(num_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DIV(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

DIV(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

EQ(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

EQ(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

EQ(num_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

EQ(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

EQ(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GE(double_ati)

Returns Boolean TRUE if the double value is greater than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GE(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GE(num_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GE(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GE(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GT(double_ati)

Returns Boolean TRUE if the double value is greater than the

argument.

Parameters:

i -

```
Double
```

Returns: bool_at

GT(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GT(num_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GT(time_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

GT(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is greater than the

argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LE(double_ati)

Returns Boolean TRUE if the double value is lesser than or

equal to the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LE(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LE(num_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LE(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LE(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than or

equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LSHIFT(uint i)

Return the Unsigned Long Value after bitwise left shifting the

current number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Left Shift Length
```

Returns: unsigned_long_at

LT(double_ati)

Returns Boolean TRUE if the double value is lesser than

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

LT(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LT(num_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LT(time_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

LT(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is lesser than

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

MAX(double_ati)

Return the maximum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MAX(ip_address_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MAX(num_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MAX(time_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MAX(unsigned_long_ati)

Return the maximum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MIN(double_ati)

Return the minimum double value after comparing

the argument with the current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MIN(ip_address_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MIN(num_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MIN(time_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MIN(unsigned_long_ati)

Return the minimum unsigned long value after comparing the argument

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MOD(ip_address_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MOD(num_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MOD(time_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MOD(unsigned_long_ati)

Return the Unsigned Long Value that is the renaminder after dividing

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MUL(double_ati)

Return the Double Value after multiplying the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

MUL(ip_address_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MUL(num_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MUL(time_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

MUL(unsigned_long_ati)

Return the Unsigned Long Value after multiplying the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** NEG**

Return the Double Value after negating the current double

value.

Returns: double_at

NE(double_ati)

Returns Boolean TRUE if the double value is not equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

NE(ip_address_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

NE(num_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

NE(time_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

NE(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long value is not equal to

the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

RSHIFT(uint i)

Return the Unsigned Long Value after bitwise right shifting the current

number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Right Shift Length
```

Returns: unsigned_long_at

SUB(double_ati)

Return the Double Value after subtracting the argument from the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

SUB(ip_address_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

SUB(num_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

SUB(time_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

SUB(unsigned_long_ati)

Return the Unsigned Long Value after subtracting the argument from the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

TYPECAST_DOUBLE_AT

Typecast the Unsigned Long Value to a Double value.

Returns: double_at

TYPECAST_NUM_AT

Typecast the Unsigned Long Value to a Number value.

Returns: num_at

TYPECAST_TEXT_T

Typecast the Unsigned Long Value to its string representation.

Returns: text_t

*** TYPECAST_UNSIGNED_LONG_AT**

Typecast the Double Value to a Unsigned Long value.

Returns: unsigned_long_at

UNSIGNED16_STRING(endian_eendianness)

Create an unsigned 16-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

UNSIGNED32_STRING(endian_eendianness)

Create an unsigned 32-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

UNSIGNED64_STRING(endian_eendianness)

Create an unsigned 64-bit binary string from the number.

Parameters(expressions not allowed):

endianness -

```
Select little or big endian
```

Returns: text_t

UNSIGNED8_STRING

Create an unsigned 8-bit binary string from the number.

Returns: text_t

UNSIGNED_LONG_FORMAT_E

Unsigned Long Format Enum

This topic lists the expressions that are provided by this class.

Enumerations

DECIMAL

Decimal representation of a positive or a negative integer. White spaces before and after the number are OK

DECIMAL_PREFIX

Decimal representation of a positive or a negative integer followed by arbitrary characters Examples " -123" " 234 " "- 123abc" "234 abc"

HEX

Hexadecimal representation of a positive or a negative integer. White spaces before and after the number are OK

HEX_PREFIX

Hexadecimal representation of a positive integer followed by arbitrary characters Examples "f" "fxyz" " f ff" all represent the number 15.

UNSIGNED_LONG_T

Provides operations to treat the underlying text as a Unsigned Long

either in the DECIMAL or the HEX format.

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, unsigned_long_at.

*** ABS**

Return absolute double value of the double value.

Returns: double_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

unsigned long.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Unsigned Long Value after bitwise negating the current

unsigned long value.

Returns: unsigned_long_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LSHIFT**(uint i)

Return the Unsigned Long Value after bitwise left shifting the

current number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>LT</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Results in Boolean TRUE value if the Text object is less than
the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the maximum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
```

```
p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Compares the string argument with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the minimum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value that is the renaminder after
dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value that is the renaminder
after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value after multiplying the argument to
the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after multiplying the
argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Double Value
after negating the current double
```

value.

Returns: double_at

*** NE**(text_ts)

Results in Boolean TRUE value if the Text object is not equal to the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is not equal to
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted cannot be longer than the public key length minus 11 bytes for padding. For example, for a 1024-bit (128 byte) key, the input text cannot be longer than 128 - 11 = 117 bytes. An Undef error will occur if the input text is too long. A null input will also cause an Undef error. Returns the (unencoded binary) encrypted text, which is typically base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +
```

```
"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Unsigned Long Value after bitwise right shifting the current
```

number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value after subtracting the argument
from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after subtracting the
argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Unsigned Long Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Typecast the Unsigned Long Value to a Number value.

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
unsigned_long_t to text_t. Will raise an undef if the value is null,
empty, or the value has an invalid format for the destination type.
(text_t : Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
TYPECAST_UNSIGNED_LONG_AT

Convert unsigned_long_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```


the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

```
    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)


returns the &lt;title&gt; element in the document's


&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the
```

```
  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }
```

```
      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{


       "creator":{


          "person":{
```

```
            "name":'Milton'

          }

        },

        "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the

    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                    XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

UNUM_TEXT_T

Provides operations to treat the underlying OctectString as Binary

String or unsigned long value .

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t, unsigned_long_at.

*** ABS**

Return absolute double value of the double value.

Returns: double_at

*** ADD**(double_ati)

Return the Double Value after adding the argument to the

current double value.

Parameters:

i -

```
Double
```

Returns: double_at

*** ADD**(ip_address_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(num_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(time_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ADD**(unsigned_long_ati)

Return the Unsigned Long after adding the argument to the

current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

> the MAC address to append.

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the string representation of the current

unsigned long.

Parameters:

str -

> the string to append.

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

> the time value to append.

Returns: text_t

*** AS_MICROSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.ffffff GMT

```
ffffff represents microseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568419 GMT

Returns: text_t

*** AS_MILLISEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fff GMT

```
fff represents milliseconds
```

Examples:

Mon, 05 Mar 2018 10:23:36.568 GMT

Returns: text_t

*** AS_NANOSEC_STR**

Returns the current system time in the following format

Format: Wday, day Month year hh:mm:ss.fffffffff GMT

```
fffffffff represents nanoseconds
```

Accuracy may be at larger granularity than nanosecond resolution.

Examples:

Mon, 05 Mar 2018 10:23:36.568419567 GMT

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BETWEEN**(double_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, ip_address_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, num_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, time_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(double_ati, unsigned_long_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(ip_address_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(num_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(time_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, double_atj)

Returns Boolean TRUE if the double value is greater than or

equal to the Lower argument and the double value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, ip_address_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, num_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, time_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BETWEEN**(unsigned_long_ati, unsigned_long_atj)

Returns Boolean TRUE if the unsigned long value is greater than or

equal to the Lower argument and the unsigned long value is lesser

than or equal to the Upper argument.

Parameters:

i -

```
Lower
```

j -

```
Upper
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BITAND**(ip_address_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(num_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(time_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITAND**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ANDing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITNEG**

Return the Unsigned Long Value after bitwise negating the current

unsigned long value.

Returns: unsigned_long_at

*** BITOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(num_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(time_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise ORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(ip_address_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(num_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(time_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BITXOR**(unsigned_long_ati)

Return the Unsigned Long Value after bitwise XORing the argument to

the current unsigned long value.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** DIV**(double_ati)

Return the Double Value after dividing the current double

value by the argument.

Parameters:

i -

```
Double
```

Returns: double_at

*** DIV**(ip_address_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(num_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(time_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** DIV**(unsigned_long_ati)

Return the Unsigned Long Value after dividing the current unsigned

long value by the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: unsigned_long_at

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EQ**(double_ati)

Returns Boolean TRUE if the double value is equal to

the argument.

Parameters:

i -

```
Double
```

Returns: bool_at

*** EQ**(ip_address_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(num_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(time_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** EQ**(unsigned_long_ati)

Returns Boolean TRUE if the unsigned long is equal to the argument.

Parameters:

i -

```
Unsigned Long
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is greater than the
```

argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is greater than the
```

argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>HASH</b></
p><div><p>Returns a positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at
*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at
*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at
*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at
*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.

Returns: bool_at

*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
```

```
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than or
```

equal to the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LSHIFT**(uint i)

Return the Unsigned Long Value after bitwise left shifting the

current number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Left Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>LT</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Results in Boolean TRUE value if the Text object is less than
the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is lesser than
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>LT</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is lesser than
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING_DEFAULT_TO_KEY</b>(<a href="text_t.html">text_t</
a>string_map_name)</p><div><p>Returns the value corresponding to the key
in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the maximum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MAX</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="num_at.html">num_at</a>i)</
```

```
p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the maximum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MAX</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
maximum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="text_t.html">text_t</a>s)</
p><div><p>Compares the string argument with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the minimum double
value after comparing
```

the argument with the current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the minimum unsigned long value after comparing the
argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MIN</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
minimum unsigned long value after comparing the argument
```

value to the current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value that is the renaminder after
dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>MOD</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value that is the renaminder
after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MOD</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value that is the renaminder after dividing
```

the current unsigned long value by the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="double_at.html">double_at</
a>i)</p><div><p>Return the Double Value after multiplying the argument to
the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>MUL</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value after multiplying the argument to
the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after multiplying the
argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>MUL</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after multiplying the argument to the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></
div><div><div><p><b>* </b><b>NEG</b></p><div><p>Return the Double Value
after negating the current double
```

value.

Returns: double_at

*** NE**(text_ts)

Results in Boolean TRUE value if the Text object is not equal to the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Returns Boolean TRUE if
the double value is not equal to
```

the argument.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Returns Boolean
TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="num_at.html">num_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="time_at.html">time_at</a>i)</p><div><p>Returns Boolean TRUE if the
unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Returns
Boolean TRUE if the unsigned long value is not equal to
```

the argument.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +
```

```
"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string and the RSA_PKCS1_PADDING padding method. The input text to be encrypted cannot be longer than the public key length minus 11 bytes for padding. For example, for a 1024-bit (128 byte) key, the input text cannot be longer than 128 - 11 = 117 bytes. An Undef error will occur if the input text is too long. A null input will also cause an Undef error. Returns the (unencoded binary) encrypted text, which is typically base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +
```

```
"MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

"CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

"79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

"-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

> identifies the configured certKey object.

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>RSHIFT</b>(uint i)</p><div><p>Return the
Unsigned Long Value after bitwise right shifting the current
```

number value by the argument number of bits.

Note that the number of bits shifted is i modulo 64.

Parameters:

i -

```
Right Shift Length</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="unum_text_t.html">unum_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="unum_text_t.html">unum_text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="double_at.html">double_at</a>i)</p><div><p>Return the Double Value
after subtracting the argument from the
```

current double value.

Parameters:

i -

```
Double</p></div><p><b>Returns: </b><a href="double_at.html">double_at</
a></p></div></div></div><div><div><p><b>* </b><b>SUB</b>(<a
href="ip_address_at.html">ip_address_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="num_at.html">num_at</a>i)</
p><div><p>Return the Unsigned Long Value after subtracting the argument
from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a href="time_at.html">time_at</
a>i)</p><div><p>Return the Unsigned Long Value after subtracting the
argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUB</b>(<a
href="unsigned_long_at.html">unsigned_long_at</a>i)</p><div><p>Return the
Unsigned Long Value after subtracting the argument from the
```

current unsigned long value.

Parameters:

i -

```
Unsigned Long</p></div><p><b>Returns: </b><a
href="unsigned_long_at.html">unsigned_long_at</a></p></div></div></
div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</p><div><p>Select
a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Typecast the Unsigned Long Value to a Double value.

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

```
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when


A and/or B is an IPv6 address, the only operator allowed between


them is '+'.


<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.


Let the server destination IPv4 address be 10.100.10.100.


Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should


produce the output "ABCD:1234::ABCD10.100.10.100".


<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result


ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Typecast the Unsigned Long Value to a Number value.

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
unum_text_t to text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
TYPECAST_UNSIGNED_LONG_AT

Convert unum_text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
                        Character</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>UNSIGNED16_STRING</b>(<a href="endian_e.html">endian_e</
a>endianness)</p><div><p>Create an unsigned 16-bit binary string from the
number.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>endianness -
```

```
                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED32_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
32-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED64_STRING</b>(<a
href="endian_e.html">endian_e</a>endianness)</p><div><p>Create an unsigned
64-bit binary string from the number.</p><p><b>Parameters</b>(expressions
not allowed):</p><div><p>endianness -



                        Select little or big endian</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>UNSIGNED8_STRING</b></p><div><p>Create an
unsigned 8-bit binary string from the number.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>URLSET_MATCHES_ANY</b>(<a href="text_t.html">text_t</a>urlset_name)</
p><div><p>Results in Boolean TRUE value if and only if the Text matches
any of
```


the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

```
    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.
Returns: text_t
*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)


returns the &lt;title&gt; element in the document's


&lt;head&gt; This can be used in a rewrite action
```

```
  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the
```

```
  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'

         }

      }
```

```
      The original JSON document is not disturbed but the



      part of the JSON body selected by the



      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

        },

        "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the

    document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                    XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },


        "title":'Paradise Lost',
```

```
        "price":"36"


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)


      This will convert the string '36' string to double if it is


      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)
```

```
      This will select "all the nodes enclosed by /Book"



      in the first 1000 bytes of the body.



      Since this is a node-set the corresponding string



      value will be returned which is "MiltonParadise



      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,
```

```
      will select "all the nodes enclosed by


      /Book/creator"


      in the first 1000 bytes of the body which is


      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the


      part of the XML body selected by the


      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:


  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
```

```
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set



      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

XPath Expression Returns: text_t </body></html>

URL_CATEGORY_T

Retrieves URL category and attributes from an URL

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from text_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

CATEGORY

Returns the category string of the URL.

If the URL does not have a category, or if the URL is malformed, this

value will be "Uncategorized".

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>GROUP</b></p><div><p>Returns the
category group of the URL. This is a higher level grouping
```

of various category strings.

If the URL does not have a category, or if the URL is malformed, this

value will be "Uncategorized".

Returns: text_t

*** GT**(text_ts)

Results in Boolean TRUE value if the Text object is greater than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"
```

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

```
    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** IS_BOOL_AT**

Results in Boolean TRUE value if the Text is of boolean type.

Returns: bool_at
*** IS_DOUBLE_AT**

Results in Boolean TRUE value if the Text is of double data type.

Returns: bool_at

*** IS_IPV6_AT**

Results in Boolean TRUE value if the Text is of IPv6 data type.

Returns: bool_at

*** IS_IPV6_SUBNET_AT**

Results in Boolean TRUE value if the Text is of IPv6 subnet data type.

Returns: bool_at

*** IS_IP_AT**

Results in Boolean TRUE value if the Text is of IPv4 data type.

Returns: bool_at

*** IS_MAC_ADDRESS_AT**

Results in Boolean TRUE value if the Text is of mac address data type.

Returns: bool_at

*** IS_NUM_AT**

Results in Boolean TRUE value if the Text is of number type.

Returns: bool_at

*** IS_NUM_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of number type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
```

```
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.
Returns: bool_at
*** LENGTH**

Results in a Number Value that is the number of characters

(not bytes) in the Text object.

Returns: num_at
*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function
and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/
oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/
g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.
Returns: text_t
*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyVAlue&gt; element with
```

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

```
the binary signature.
```

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.

Returns: bool_at

*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;
```

```
 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -



                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>REPUTATION</b></p><div><p>Returns the reputation score
of the URL. Where a score 5 is most risky
```

and 1 is least risky.

If the URL does not have a category, or if the URL is malformed, this

value will be -1.

Returns: num_t

*** SET_CHAR_SET**(charset_echarset)

Set the character set to use for subsequent functions that

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="url_category_t.html">url_category_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

WW denotes the ascii value of character Y in hexadecimal and HH denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//" will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in the Text object. No percent decoding will be performed on the Text object.

PLUS_AS_SPACE - If set then all Text operations are carried out after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the ascii value of character Y in octal. The valid range of octal values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

denotes the ascii value of character Y in hexadecimal and XX

denotes the ascii value of character Z in hexadecimal.

For example, the encoded text "http%u3a2f/" and "http%u003a//"

will both be decoded to"http://"

4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="url_category_t.html">url_category_t</a></p></div></div></
div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</p><div><p>Select
the Text after skipping over a prefix of the current Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t
*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t
*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
```

```
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
```

(http_cache_control_t : This object provides for the ability to work on HTTP Cache-Control headers.) Returns: http_cache_control_t * TYPECAST_HTTP_CONTENT_LENGTH_T Convert text_t to http_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or HTTP.RES.CONTENT_LENGTH object can be

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t
*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t
*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t

*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t

*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t

*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the


concatenation of the string representations of the individual PIXL


expressions.  No other operator other than '+' is expected to work


when any of the operands is an IPv6 PIXL expression.  Also, IPv6


PIXL expressions can be combined with any other type of PIXL


expression except a PIXL expression that produces a bool result.


In other words, if A and/or B are two IPv6 addresses, A+B should result


in the string AB.  When one of A or B is an IPv6 address, the other
```

```
can only be a number or a string and not a boolean.  Further, when



A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t
*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
```

```
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be

treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t
*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t
*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t
*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t
*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>TYPECAST_TEXT_T</b></p><div><p>Convert
url_category_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
```

```
an undef if the value is null, empty, or the value has an invalid format
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of
```

```
  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'
```

```
          }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{
```

```
            person:{


              name:'Milton'


            }


          }


       The original JSON document is not disturbed but the


       part of the JSON body selected by the


       expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                          XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:
```

```
{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)
```

```
      This will return true as there is a node named



      'creator' with parent 'Book' in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



       "creator":{



          "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

 }

 HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

     This will convert the string '36' string to double if it is

     in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                       XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }


        },
```

```
        "title":'Paradise Lost'


     }


  }


  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


      This will select "all the nodes enclosed by /Book"


      in the first 1000 bytes of the body.


      Since this is a node-set the corresponding string


      value will be returned which is "MiltonParadise


      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the



      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select
```

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton"</
p> Parameters(expressions not allowed): xpathex -

XPath Expression Returns: <a
href="text_t.html">text_t </body></html>

VENDOR_SPEC_APPID_AVP_T

Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

For example, to retrieve the Auth-Application-Id AVP from the

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

This topic lists the expressions that are provided by this class.

Note: Expressions with the *****symbol are inherited/promoted from avp_t.

ACCT_APPLICATION_ID

Returns the Acct-Application-Id AVP (code 259) that is nested inside the

Vendor_Specific_Application_Id AVP.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select



"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

AUTH_APPLICATION_ID

Returns the Auth-Application-Id AVP (code 258) that is nested inside the

Vendor_Specific_Application_Id AVP.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t

*** AVPLIST_AFTER**

Returns the region after this AVP in diameter message (or in other words

returns location at end of this AVP and length covering end

of this AVP to end of diameter message.

If this AVP is inside a Grouped AVP, this method returns region after

this AVP in parent Grouped AVP.

Returns: text_t

*** AVPLIST_BEFORE**

Returns the region till this AVP in diameter message (or in other words

returns location at start of AVPs and length equal to offset of

this AVP).

If this AVP is inside a Grouped AVP, this method returns region till

this AVP in parent AVP.

Returns: text_t

*** AVP**(uint avpcode)

Returns the AVP nested inside the GROUPED AVP (the Data field is a

sequence of AVPs).

For example, to retrieve AVP(263) that is nested in AVP(99999) in a

request, the expression is: DIAMETER.REQ.AVP(999999).AVP(263).

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

Returns: avp_t

*** AVP**(uint avpcode, uint vendorid)

Returns the specified AVP in given vendor-id namespace and that is

nested inside a GROUPED AVP in the message.

For example, to get the AVP with code 1005 and VendorId 10415 from a Request,

that is nested inside 1001 (vendor-id 10415). Use this expression:

DIAMETER.REQ.AVP(1001, 10415).AVP(1004, 10415).

Note: To get AVP that is nested inside a grouped AVP, use the

AVP method from the avp_t class.

Parameters(expressions not allowed):

avpcode -

```
AVP code
```

vendorid -

```
Vendor Id
```

Returns: avp_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Returns the number of occurences of an AVP in diameter message.

The system does not keep track of more than 16 AVP occurences of an AVP. The result

will be undefined if there are more than 16 instances of that AVP.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns true if the AVP exists in the message.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.
Returns: text_t
*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t
*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t
*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

Based on RFC2396, following are considered URL safe characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Exclamation mark: "!"

Tilde: "~"

Asterix: "*"

Single quote: "'"

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

```
    At symbol: "@"


    Question mark: "?"


    Equals: "="


    Dollar: "$"


    Percent: "%"


    Ampersand: ";"


    Plus: "+"


    Comma: ","


    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t

*** INSTANCE**(uint instanceno)

Returns instances of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

INSTANCE(0) retrieves the first instance, INSTANCE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="avp_t.html">avp_t</a></p></div></div></div><div><div><p><b>* </
b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of boolean type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_DOUBLE_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of double data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_IPV6_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 data type.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of IPv6 subnet data type.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_IP_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of IPv4 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_MAC_ADDRESS_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of mac address data type.</p><p><b>Returns: </b><a
```

```
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_MANDATORY</b></p><div><p>Returns true if the AVP has the Mandatory
bit set.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></
div></div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results
in Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </
b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>IS_NUM_AT</b>(<a
href="num_format_e.html">num_format_e</a>format)</p><div><p>Results in
Boolean TRUE value if the Text is of number type
```


with given input base.

Parameters(expressions not allowed):

format -


```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```


present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -


```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_VENDOR_SPECIFIC</b></
p><div><p>Returns true if the AVP has Vendor-Specific bit (V) set.</
p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Returns the AVP length including AVP header.

If the AVP is not present, it returns length 0. If multiple occurences

of the AVP exist, it returns the length of first occurence.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

> a text expression that supplies a RSA public key in the

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +


 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +


 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +


 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +


 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyVAlue> element with

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

> the binary signature.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -




                       Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="vendor_spec_appid_avp_t.html">vendor_spec_appid_avp_t</a></p></
div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```


many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="vendor_spec_appid_avp_t.html">vendor_spec_appid_avp_t</a></p></
div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
```

```
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>TYPECAST_AVP_T</b></
p><div><p>Convert vendor_spec_appid_avp_t to avp_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (avp_t : Operations that can be performed on the
returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t

*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -
```

```
                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
```

```
the value is null, empty, or the value has an invalid format for the
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t
*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t
*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert text_t to
radius_avp_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (radius_avp_t :
Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t
*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
avp_t to text_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (text_t : Designates
any piece of text.)</p><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TIME_AT</b></
p><div><p>Convert text_t to time_at. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(time_at : Treats a number value as number of seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
```

```
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```

values and retains the rest. This is a read only operation.

```
Based on RFC3986, following are considered URL unreserved characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Tilde: "~"
```

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t
*** VALUE**

Returns the value of first instance of the AVP as OctetString..

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com

- Route-Record: abcd3.netscaler.com

Alternatively, you can use the expression: DIAMETER.REQ.AVP(282).

Returns: text_t
*** VALUE**(uint instanceno)

Returns value of an instance of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: DIAMETER.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

VALUE(0) retrieves the first instance, VALUE(1) retrieves the second

instance, and so on.

For example, lets assume the Route-Record AVP (code 282) has three

instances in a request.

- Route-Record: abcd1.netscaler.com
- Route-Record: abcd2.netscaler.com
- Route-Record: abcd3.netscaler.com

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>VENDORID</b></p><div><p>Returns the Vendor-Id AVP
(code 266) that is nested inside the
```

Vendor_Specific_Application_Id AVP.

Note:expression can be used to retrieve the value as Unsigned32

Returns: avp_unum_t
*** VENDOR_ID**

Returns the Vendor ID if the AVP has the Vendor-Specific bit set.

Returns: unsigned_long_at

*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)



  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child
```

```
  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{
```

```
            "name":'Milton'

          }

       },

       "title":'Paradise Lost'

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is
```

```
        creator:{

           person:{

             name:'Milton'

           }

        }

      The original JSON document is not disturbed but the

      part of the JSON body selected by the

      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}
```

```
  HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

      This will return true as there is a node named

      'creator' with parent 'Book' in the first 1000 bytes of the

      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -

                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:

  {

     "Book":{

        "creator":{
```

```
          "person":{

            "name":'Milton'

          }

       },

       "title":'Paradise Lost',

       "price":"36"

    }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

    This will convert the string '36' string to double if it is

    in the first 1000 bytes of the document.</p><p><b>Parameters</
```

```
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the JSON file:


  {


     "Book":{


        "creator":{


           "person":{


             "name":'Milton'


           }
```

```
        },

        "title":'Paradise Lost'

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)

      This will select "all the nodes enclosed by /Book"

      in the first 1000 bytes of the body.

      Since this is a node-set the corresponding string

      value will be returned which is "MiltonParadise

      Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -
```

```
                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)



      This expression when operated on the above XML file,



      will select "all the nodes enclosed by



      /Book/creator"



      in the first 1000 bytes of the body which is



      &lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/person>&lt;/
creator>The original XML document is not disturbed but the



      part of the XML body selected by the
```

```
      xpath expression is marked for further processing.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath operates on XML file and
```

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
```

```
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)



      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:
```

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)

This expression will select

"all the nodes enclosed by /Book/creator"

in the first 1000 bytes of the body. Since this is a node-set

its string value is returned which is: "Milton" Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t </body></html>

VENDOR_SPEC_RADIUS_AVP_T

- Vendor-Id

To retrieve the Vendor-Id from the

Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

This topic lists the expressions that are provided by this class.

Note: Expressions with the ***symbol are inherited/promoted from radius_avp_t.

*** AFTER_REGEX**(re regex)

Selects the text after the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

Example:

```
HTTP.REQ.HEADER("etag").AFTER_REGEX(re/KX/) will select


"DAPPER" from "KXDAPPER".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** AFTER_STR_ANY**(text_tpatset_name)

Selects the Text that is present after the first occurrence

of any of the strings that are bound to the patset named by

the string argument. If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** AFTER_STR**(text_ts)

Selects the Text that is present after the first occurrence

of the string argument. If there is no match for the

argument in the current Text then the result Text object has

0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** APPEND**(bool_atb)

Appends the given boolean's string representation to the

current string.

Parameters:

b -

```
the boolean value to append.
```

Returns: text_t

*** APPEND**(double_ati)

Appends the given double's string representation to the current

string.

Parameters:

i -

```
the double value to append.
```

Returns: text_t

*** APPEND**(ip_address_ataddr)

Appends the given IPv4's string representation to the current string.

Parameters:

addr -

```
the IPv4 address to append.
```

Returns: text_t

*** APPEND**(num_ati)

Appends the given number's string representation to the current

string.

Parameters:

i -

```
the number value to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** APPEND**(unsigned_long_ati)

Appends the given unsigned long's string representation to the

current string.

Parameters:

i -

```
the unsigned long value to append.
```

Returns: text_t

*** APPEND**(ipv6_address_ataddr)

Appends the given IPv6's string representation to the current string.

Parameters:

addr -

```
the IPv6 address to append.
```

Returns: text_t

*** APPEND**(ipv6_subnet_ataddr)

Appends the given IPv6 subnet's string representation to the current string.

Parameters:

addr -

```
the IPv6 subnet to append.
```

Returns: text_t

*** APPEND**(mac_address_ataddr)

Appends the given MAC's string representation to the current string.

Parameters:

addr -

```
the MAC address to append.
```

Returns: text_t

*** APPEND**(text_tstr)

Appends the given string to the current string.

Parameters:

str -

```
the string to append.
```

Returns: text_t

*** APPEND**(time_att)

Appends the given time's string representation to the current string.

Parameters:

t -

```
the time value to append.
```

Returns: text_t

*** AVPLIST_AFTER**

Returns the region after this AVP in radius message (or in other words

returns location at end of this AVP and length covering end

of this AVP to end of radius message.

Returns: text_t

*** AVPLIST_BEFORE**

Returns the region till this AVP in radius message (or in other words

returns location at start of AVPs and length equal to offset of

this AVP).

Returns: text_t

*** B64DECODE**

Decodes the selected text using base64 decoding.

For example, the string "TWFu" is converted to "Man"

Returns: text_t

*** B64ENCODE**

Encodes the selected text using base64 encoding.

For example, the string "Man" is converted to "TWFu"

Returns: text_t

*** BEFORE_REGEX**(re regex)

Selects the text before the portion that matches the given

regular expression argument. The first match is returned

when multiple matches occur in the text. The match may occur

anywhere within the text - if you wish to "anchor" this,

use "\^" at the start of the regex and/or "$" at the end of

the regex. When the regular expression does not match the

given string, a Text object of length of 0 is returned.

For example:

```
HTTP.RES.HEADER("content-type").BEFORE_REGEX(re#/#) will select "text"
from "text/plain".
```

Parameters(expressions not allowed):

regex -

```
Regular Expression
```

Returns: text_t

*** BEFORE_STR_ANY**(text_tpatset_name)

Selects the Text that is present before the first occurrence

of any of the strings that are bound to the patset or

datasetnamed by the string argument.

If there is no match to any of the strings

bound to the patset in the current Text object then the result

Text object has 0 length.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: text_t

*** BEFORE_STR**(text_ts)

Selects the Text that is present before the first occurrence

of the string argument. If there is no match for the

argument in the current Text object then the result Text

object has 0 length.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: text_t

*** BETWEEN**(text_tstart, text_tend)

Results in Boolean TRUE value if length of the Text object

is greater than or equal to the sum of the Start and End argument

lengths and if the prefix of the Text object

matches the Start string argument and if the Suffix of the Text

object matches the End string argument.

Parameters:

start -

```
Start
```

end -

```
End
```

Returns: bool_at

*** BIN_DECODE**(binary_encoding_eencoding)

Decode the encoded input into its original byte sequence. An incorrectly

enooded input will result in an Undef error.

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are encoded.
```

Returns: text_t

*** BIN_ENCODE**(binary_encoding_eencoding)

Encode the input byte sequence. Note that BIN_ENCODE(hex_colons) is

equivalent to BLOB_TO_HEX().

Parameters(expressions not allowed):

encoding -

```
specifies how the bytes are to be encoded.
```

Returns: text_t

*** BLOB_TO_HEX**

Convert to hexadecimal representation and select text.

For example, the byte string abc is converted to 61:62:63.

Returns: text_t

*** COMPARE**(text_ts)

Results in a Number Value which is the result of comparison of

the input argument with the Text object. The result is

greater than 0 if the Text object is lexicographically

greater than the argument. The result is 0 if the Text

object and the argument match. The result is less than 0 if

the Text object is lexicographically lesser than the argument.

Parameters:

s -

```
String
```

Returns: num_at

*** CONTAINS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text contains any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** CONTAINS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

contains any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no

match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** CONTAINS**(text_ts)

Results in Boolean TRUE value if the Text contains the

string given in the argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String
```

Returns: bool_at

*** COUNT**

Returns the number of occurences of an AVP in radius message.

The system does not keep track of more than 16 AVP occurences of an AVP. The result

will be undefined if there are more than 16 instances of that AVP.

Returns: num_at

*** CRC32**

Returns a 32-bit integer computed by applying a

CRC (cyclic redundancy check) function to the selected

text. This is the CRC function used by Ethernet,

MPEG-2, PKZIP, GZIP, PNG, and many other applications.

For example, the expression to compute the CRC value

of the (first 1000 bytes of the) body of an HTTP request is:

HTTP.REQ.BODY(1000).CRC32

Returns: num_at

*** DECODE_USING_TEXT_MODE**

Decodes the selected text using the currently configured

text encoding methods like URLENCODED, BACKSLASH_ENCODED,

PLUS_AS_SPACE, NOURLENCODED, NO_BACKSLASH_ENCODED and

NO_PLUS_AS_SPACE. Text encoding methods are set

using SET_TEXT_MODE method.

Returns: text_t

*** DECRYPT**

Decrypt the selected base64-encoded ciphertext using the

configured encryptionParams.

Returns: text_t

*** DECRYPT**(binary_encoding_ein_encoding)

Decrypt the selected ciphertext using the configured encryptionParams.

Parameters(expressions not allowed):

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DECRYPT**(text_tencryptionKey)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** DECRYPT**(text_tencryptionKey, binary_encoding_ein_encoding)

Decrypt the selected base64-encoded encrypted bytes using the

configured encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format,

and optional initialization vector (IV).

in_encoding -

```
specifies how the input ciphertext is encoded.
```

If the ciphertext is not encoded as specified, causes an Undef error.

Returns: text_t

*** DIGEST**(digest_method_emethod)

Returns a byte sequence computed by applying the

the specified message digest method to the selected

text.

Parameters(expressions not allowed):

method -

```
digest method to apply to the input text
```

Returns: text_t

*** ENCRYPT**

Encrypt the selected plaintext bytes using the configured

encryptionParams. The resulting ciphertext is base64-encoded.

Returns: text_t

*** ENCRYPT**(binary_encoding_eout_encoding)

Encrypt the selected plaintext bytes using the configured

encryptionParams.

Parameters(expressions not allowed):

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENCRYPT**(text_tencryptionKey)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

Returns: text_t

*** ENCRYPT**(text_tencryptionKey, binary_encoding_eout_encoding)

Encrypt the selected cleartext bytes using the configured

encryption key.

Parameters(expressions not allowed):

encryptionKey -

```
identifies the encryptionKey object
```

to provide the cipher method, mode, padding, key, encrypted format

and optional initialization vector (IV).

out_encoding -

```
specifies how the resulting ciphertext is encoded.
```

Returns: text_t

*** ENDSWITH_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text ends with any of

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** ENDSWITH_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

ends with any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** ENDSWITH**(text_tend)

Results in Boolean TRUE value if a suffix of the Text

object matches the string argument.

Parameters:

end -

```
String
```

Returns: bool_at

*** EQUALS_ANY**(text_tpatset_name)

Results in Boolean TRUE value if the Text equals any of

the strings that are bound to the patset or

dataset named by the string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: bool_at

*** EQUALS_INDEX**(text_tpatset_name)

Returns the numerical index of the matching string, if the Text

equals any of the strings that are bound to the patset or dataset

named by the string argument. Returns 0 if there is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name
```

Returns: num_at

*** EQ**(text_ts)

Results in Boolean TRUE value if the Text object matches the

input argument.

Parameters:

s -

```
String
```

Returns: bool_at

*** EXISTS**

Returns true if the AVP exists in the message.

Returns: bool_at

*** GET_SIGNED16**(uint n, endian_eendianness)

Get the 16-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED32**(uint n, endian_eendianness)

Get the 32-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_SIGNED8**(uint n)

Get the 8-bit signed binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_UNSIGNED16**(uint n, endian_eendianness)

Get the 16-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: num_at

*** GET_UNSIGNED32**(uint n, endian_eendianness)

Get the 32-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED64**(uint n, endian_eendianness)

Get the 64-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

endianness -

```
Select little or big endian
```

Returns: unsigned_long_at

*** GET_UNSIGNED8**(uint n)

Get the 8-bit unsigned binary integer at the given offset.

Parameters:

n -

```
Offset in bytes
```

Returns: num_at

*** GET_URLSET_METADATA**(text_turlset_name)

Returns metadata value corresponding to pattern in the urlset.

The matching is case-insensitive for domains, but case-sensitive

for paths. If no pattern is matched within the urlset, the return

value is a NULL string.

For example:

</p>
urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p>&lt;/div>&lt;/div>&lt;/div><div><div><p><b>* </b><b>GE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than or
```

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>GT</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is greater than the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>HASH</b></p><div><p>Returns a
positive 31 bit integer value computed
```

by applying a proprietary Citrix ADC hash function

to the selected text. This function is not case-

sensitive and ignores whitespaces. For example,

applying the HASH function to two strings "Ab c"

and "abc" would produce the same value. This

function should not be used for security purposes;

use instead the DIGEST function with one of the

standard digest methods.

Returns: num_at
*** HMAC**(text_thmacKey)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key.

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

Returns: text_t

*** HMAC**(text_thmacKey, binary_encoding_eout_encoding)

Returns a byte sequence computed by applying the HMAC

(Hash Message Authenticastion Code) function to the selected

text using the configured HMAC key .

Parameters(expressions not allowed):

hmacKey -

```
indentifies the hmacKey object to provide the
```

digest (hash) method and the key.

out_encoding -

```
specifies how the resulting HMAC is encoded.</p></div><p><b>Returns: </
b><a href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>*
</b><b>HTML_XML_SAFE</b></p><div><p>Apply XML safe transform and select
text.
```

Converts '<', '>' and ';'

in the input text to ';lt;', ';gt;' and ';amp;'

respectively; other characters in the input are not modified.

This transform safeguards against XSS based attacks.

This is a read only operation.

Returns: text_t

*** HTTP_HEADER_SAFE**

Apply Header safe transform and select text. Converts all new line

('\n') characters present in the input text to '%0A' so that they can

safely be used in HTTP headers. This transform safeguards against

response-splitting attacks. This is a read only operation.

Returns: text_t

*** HTTP_URL_SAFE**

Apply URL safe transform and select text. Converts URL unsafe

characters to corresponding '%xx' values and retains the rest.

This is a read only operation.

```
Based on RFC2396, following are considered URL safe characters:


    Alpha-numeric characters: a-z, A-Z, 0-9


    Hyphen and underscore: "-", "_"


    Dot: "."


    Exclamation mark: "!"


    Tilde: "~"


    Asterix: "*"


    Single quote: "'"
```

Open and close parentheses: "(", ")"

Semi-colon: ";"

Colon: ":"

At symbol: "@"

Question mark: "?"

Equals: "="

Dollar: "$"

Percent: "%"

Ampersand: ";"

Plus: "+"

Comma: ","

```
    Forward slash: "/"
```

representation of the input character.

If the requirement is to encode all but URL unreserved

characters, please consider using URL_RESERVED_CHARS_SAFE

method.

Returns: text_t
*** INSTANCE**(uint instanceno)

Returns instances of the AVP.

The first instance of the AVP can be retrieved by simply using the

expression: radius.REQ.AVP(26). However, subsequent instances must be

retrieved using this method.

INSTANCE(0) retrieves the first instance, INSTANCE(1) retrieves the second

instance, and so on.

For example, to get the second instance, the expression would be:
radius.REQ.AVP(282).INSTANCE(1).

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="radius_avp_t.html">radius_avp_t</a></p></div></div></
div><div><div><p><b>* </b><b>IS_BOOL_AT</b></p><div><p>Results in Boolean
TRUE value if the Text is of boolean type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_DOUBLE_AT</b></p><div><p>Results in Boolean TRUE value if the Text
is of double data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_AT</b></p><div><p>Results in Boolean TRUE value if the Text
```

```
is of IPv6 data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IPV6_SUBNET_AT</b></p><div><p>Results in Boolean TRUE value if the
Text is of IPv6 subnet data type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_IP_AT</b></p><div><p>Results in Boolean TRUE value if the Text is
of IPv4 data type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_MAC_ADDRESS_AT</b></
p><div><p>Results in Boolean TRUE value if the Text is of mac address data
type.</p><p><b>Returns: </b><a href="bool_at.html">bool_at</a></p></div></
div></div><div><div><p><b>* </b><b>IS_NUM_AT</b></p><div><p>Results in
Boolean TRUE value if the Text is of number type.</p><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_NUM_AT</b>(<a href="num_format_e.html">num_format_e</a>format)</
p><div><p>Results in Boolean TRUE value if the Text is of number type
```

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>IS_STRINGMAP_KEY</b>(<a
href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns true if
the string that it operates on is
```

present in the string map.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). IS_STRINGMAP_KEY("url_stringmap") returns true if value of HTTP.REQ.URL is present as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>IS_TIME_AT</b></p><div><p>Results in Boolean TRUE if the Text follows
any of the
```

standard time formats given below, otherwise false.

Tue, 05 Nov 1994 08:12:31 GMT RFC822

Tuesday, 05-Nov-94 08:12:31 GMT RFC850

Tue Nov 4 08:12:31 1994 asctime

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**

Results in Boolean TRUE value if the Text is of unsigned long data type.

Returns: bool_at
*** IS_UNSIGNED_LONG_AT**(num_format_eformat)

Results in Boolean TRUE value if the Text is of unsigned long data type

with given input base.

Parameters(expressions not allowed):

format -

```
number format</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>JWT_VERIFY_CERTKEY</b>(<a
href="text_t.html">text_t</a>certkey_name)</p><div><p>Verifies the
signature on the input JSON Web Token (JWT), using the RSA public key in
```

the X.509 certificate in the specified certkey object. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'

- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, if the token is not

correctly formatted, or if the verification operation cannot complete.
Returns: bool_at
*** JWT_VERIFY_XML**(text_txml_pkey)

Verifies the signature on the input JSON Web Token (JWT), using the RSA public key

supplied by the specified XML element. The token has the format

HEADER.PAYLOAD.SIGNATURE

where

- HEADER is a base64-encoded JSON structure indicating a signature algorithm and token type, e.g.'{"alg":"RS256","typ":"JWT"}'
- PAYLOAD is a base64-encoded JSON structure with asserted claims, e.g. '{"loggedInAs":"admin","iat":1422779638}'
- SIGNATURE is the base64-encoded bytes for the signature on HEADER.PAYLOAD.

public key; false if the signature is not verified

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the public key.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, if the token is not correctly formatted, or if the

verification operation cannot complete.

Returns: bool_at

*** LENGTH**

Returns the AVP length including AVP header.

If the AVP is not present, it returns length 0. If multiple occurences

of the AVP exist, it returns the length of first occurence.

Returns: num_at

*** LE**(text_ts)

Results in Boolean TRUE value if the Text object is less than or

equal to the input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>LOCATION_LONG</b></
p><div><p>Get the location qualifier from CLIENT.IP.SRC.LOCATION.GET(n)
```

(ISO short name) and then convert it to human readable long name.

This conversion is valid only for country names. Correct Country

qualifier value (number) in GET(n) is left up to the user as

qualifiers are user configurable.

Expression example: CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG

For example: Data from each expression

CLIENT.IP.SRC.LOCATION: NA.US.California.SantaClara..

CLIENT.IP.SRC.LOCATION.GET(1): US

CLIENT.IP.SRC.LOCATION.GET(1).LOCATION_LONG: United States

Returns: text_t
*** LT**(text_ts)

Results in Boolean TRUE value if the Text object is less than the

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>MAP_STRING_DEFAULT_TO_KEY</
b>(<a href="text_t.html">text_t</a>string_map_name)</p><div><p>Returns the
value corresponding to the key in the
```

string map. This is similar to the MAP_STRING method, but if

the key is not present then instead of returning a null

string, it returns the key itself.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAP_STRING</b>(<a href="text_t.html">text_t</a>string_map_name)</
p><div><p>Returns value corresponding to key in the string map.
```

Key is the string that it is given as input.

IGNORECASE, NOIGNORECASE - controls whether key

lookup in string map is case sensitive or not.

For example:

- HTTP.REQ.URL.MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap".
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE). MAP_STRING("url_stringmap") returns value when the value of HTTP.REQ.URL is used as key in the string map whose name is "url_stringmap". Key lookup in this case is done case-insensitively.

Parameters(expressions not allowed):

string_map_name -

```
identifies the string map.</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MARK_SAFE</b></p><div><p>Mark the PIXL expression safe without
applying any safe transform.</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>MAX</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Compares the
string argument with the current string and
```

returns the max string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>MIN</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Compares the string argument
with the current string and
```

returns the min string.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>NE</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Results in Boolean TRUE value
if the Text object is not equal to the
```

input argument.

Parameters:

s -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>NORMALIZE_URL</b></
p><div><p>Normalize URL.
```

Normalizes a URL by doing "percent" decoding. See the

description for URLENCODED in the SET_TEXT_MODE

function.

This performs the equivalent of

SET_TEXT_MODE(URLENCODED).DECODE_USING_TEXT_MODE,

except that it does not change the value of the

current text mode.

Returns: text_t
*** PKEY_ENCRYPT_PEM_NO_PADDING**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string

and RSA_NO_PADDING padding method. The input text to be encrypted cannot be

longer than the public key length minus 11 bytes for padding. For example,

for a 1024-bit (128 byte) key, the input text cannot be longer than

128 - 11 = 117 bytes. An Undef error will occur if the input text is

too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_ENCRYPT_PEM**(text_tpem_pkey)

Encrypts the input text using the RSA public key provided by the PEM string
and the RSA_PKCS1_PADDING padding method. The input text to be encrypted

cannot be longer than the public key length minus 11 bytes for padding.

For example, for a 1024-bit (128 byte) key, the input text cannot be

longer than 128 - 11 = 117 bytes. An Undef error will occur if the input

text is too long. A null input will also cause an Undef error.

Returns the (unencoded binary) encrypted text, which is typically

base-64 encoded for use in a message.

Parameters:

pem_pkey -

```
a text expression that supplies a RSA public key in the
```

PEM format. If the text value is not a valid PEM RSA PUBLIC KEY

encoding, an Undef error will occur. Example (a 1024-bit public key):

```
"-----BEGIN RSA PUBLIC KEY-----" +

 "MIGJAoGBAKl5vgQEj73Kxp+9yn1v5gPR1pnc4oLM2a0kaWwBOsB6rzCIy6znwnvw" +

 "CY1xRvQhRlJSAyJbloL7wZFIJ2FOR8Cz+8ZQWXU2syG+udi4EnWqLgFYowF9zK+o" +

 "79az597eNPAjsHZ/C2oL/+6qY5a/f1z8bQPrHC4GpFfAEJhh/+NnAgMBAAE=" +

 "-----END RSA PUBLIC KEY-----"
```

Returns: text_t

*** PKEY_SIGN_CERTKEY**(text_tcertkey_name)

Generates a binary digital signature using the SHA-256 message digest function and an RSA private key from the specified SSL certKey object.

Parameters:

certkey_name -

```
identifies the configured certKey object.
```

Results in an Undef error if the certkey object does not exist or does not

contain an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_SIGN_XML**(text_txml_pkey)

Generates a binary digital signature using the SHA-256 message digest function

and an RSA private key from the specified XML element.

Parameters:

xml_pkey -

```
an expression that supplies an &lt;RSAKeyValue&gt; element with
```

the private key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

<P>08udZRun1Ts4Fs7ctk8znvr6LEoqIDaAI8GpmE8MXF9OCr0Z3847kNd/
loKFgRZSV4KABJPu6T2l8Pr1xma8iQ==</P>

<Q>zNjysxcIJV/J9cTpIgPp2/
x1IXHT5GTH8OMuNad0qtRSLjU1QYUmmkm9QcJJAN+7h8FEXWGabySUzBrHj/IEbw==</Q>

<DP>QavUDQLLQpnPJXtVAo6CYhxOMCd68mwTxYKKNLjmC0pd54SFHllTLPE1FZU5EjK6q3nCeijonpfH+XX
DP>

<DQ>EiXOKouJzzXTQJC3TqomF7Uf81yQNxWumOqnd11kepXgyfMwyqMvLlAGEkNSEF/
Bz18R3Wv+MIZ/IC3nLBt2xQ==</DQ>

<InverseQ>ocuCmz36CP+N2aGqxaU+BKD5+3RLgRq9LFhULEz2jKx0iV9r76jsAxn3sLcTYYO9eqRPy2cz74AU
nw==</InverseQ>

<D>fTBD773a3GzcmgVdfZlaBLpN1ux2mf5uUVrTUQf7eVFm9egm4/tzC5A9zjC+I/oscwTH8epL3UfRLdtp05Q6pWfJBXEnGo0QmOFFBK4pLx/5JJpPGZ69ovDFEVOUHyiCfKE/g97T4WX3us0nFe1272zwGlHarM1YYwvfeE08UdE=</D>

</RSAKeyValue>

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA private key, or if the signing operation cannot complete.

Returns: text_t

*** PKEY_VERIFY_CERTKEY**(text_tcertkey_name, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the specified

RSA public key in the X.509 certificate in the specified certKey object.

Returns true if the signature is verified, false otherwise.

Parameters:

certkey_name -

```
identifies the certkey object containing an X.509 certificate
```

holding the RSA public key.

signature -

```
the binary signature.
```

Results in an Undef error if the certkey object does not exist or does not

contain an X.509 certificate with an RSA public key, or if the verification

operation cannot complete.

Returns: bool_at

*** PKEY_VERIFY_XML**(text_txml_pkey, text_tsignature)

Verifies if the input text matches the specified signature generated using the

SHA-256 messge digest function and the RSA private key associated with the RSA

public key supplied by the specified XML element. Returns true if the signature

is verified, false otherwise.

Parameters:

xml_pkey -

> an expression that supplies an <RSAKeyVAlue> element with

the public key. Example:

<RSAKeyValue>

<Modulus>qXm+BASPvcrGn73KfW/mA9HWmdzigszZrSRpbAE6wHqvMIjLrOfCe/
AJjXFG9CFGUlIDIluWgvvBkUgnYU5HwLP7xlBZdTazIb652LgSdaouAVijAX3Mr6jv1rPn3t408COwdn8Lagv/
7qpjlr9/XPxtA+scLgakV8AQmGH/42c=</Modulus>

<Exponent>AQAB</Exponent>

</RSAKeyValue>

signature -

> the binary signature.

Results in an Undef error if the XML element is invalid or does not correctly

specify an RSA public key, or if the verification operation cannot complete.
Returns: bool_at
*** PREFIX**(char c, uint i)

Select a prefix of the Text. The selected prefix is the

longest prefix that has at most Count (2nd argument) occurrences

of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>PREFIX</b>(uint i)</p><div><p>Select a prefix of the Text. The length
of the
```

prefix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>REGEX_MATCH</b>(re
regex)</p><div><p>Returns TRUE if and only if the text contains at least
one match
```

of the regular expression argument. The match may occur anywhere

within the text - if you wish to "anchor" this, use "\^" at the

start of the regex and/or "$" at the end of the regex.

The regular expression has to be of the format:

```
 <code>re&lt;delimiter&gt;regular expression&lt;same delimiter&gt;



 </code><br /><br /><br /><br />The regular expression cannot be bigger
than 1499 characters.
```

The regular expression should be acceptable by the PCRE regular

expression library. Look at http://www.pcre.org/pcre.txt for

documentation on the regular expression syntax. In particular,

look at the pcrepattern man page.

There are some differences in the allowed syntax w.r.t PCRE syntax.

Backreferences are not allowed. It is recommended not to use

recursive regular expressions. The dot metacharacter matches

newlines also. Unicode is not supported. SET_TEXT_MODE(IGNORECASE)

overrides the (?i) internal option specified in the regular

expression.

The PCRE library configuration can be seen in the system log, after

executing 'nsapimgr -B" call ns_pixl_regex_show_config()"' at the

shell prompt.

Some examples:

- HTTP.REQ.HOSTNAME.REGEX_MATCH(re/[[:alpha:]]+(abc){2,3}/)
- HTTP.REQ.URL.SET_TEXT_MODE(URLENCODED).REGEX_MATCH(re#(a*b+c*)#)
- HTTP.REQ.URL.REGEX_MATCH(re/a(?i)b/) matches ab and aB
- HTTP.REQ.URL.SET_TEXT_MODE(IGNORECASE).REGEX_MATCH(re/ab/) matches ab, aB, Ab and AB
- HTTP.REQ.BODY.REGEX_MATCH(re/(?ixm) (\^ab (.*) cd$) /) does a case-insensitive, multiline match where the dot metacharacter matches newline also. The regex itself can contain whitespace which will be ignored during the matching (x - extended regex).

Parameters(expressions not allowed):

regex -

```
Regular Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>REGEX_SELECT</b>(re regex)</p><div><p>Selects the text that matches
the given regular expression
```

argument. The first match is returned when multiple matches

occur in the text. The match may occur anywhere within the

text - if you wish to "anchor" this, use "\^" at the start

of the regex and/or "$" at the end of the regex. When the

regular expression does not match the given string, a Text

object of length of 0 is returned.

For example:

```
 HTTP.REQ.HEADER("via").REGEX_SELECT(re!NS-CACHE-\d\.\d:\s*\d{1,3}!) will
select "NS-CACHE-7.0:  70"</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>regex -




                        Regular Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>REPEAT</b>(uint i)</p><div><p>Produces a string which contains 'i'
sequential copies of the input
```

string.

Parameters:

i -

```
the number of times to repeat the string (0 or more)</p></
div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>SET_CHAR_SET</b>(<a
href="charset_e.html">charset_e</a>charset)</p><div><p>Set the character
set to use for subsequent functions that
```

use text.

Parameters(expressions not allowed):

charset -

```
Character Set</p></div><p><b>Returns: </b><a
href="vendor_spec_radius_avp_t.html">vendor_spec_radius_avp_t</a></p></
div></div></div><div><div><p><b>* </b><b>SET_TEXT_MODE</b>(<a
href="textmode_e.html">textmode_e</a>mode)</p><div><p>The mode setting of
a Text object controls the behavior of
```

many Text operations. The Text mode settings may also

control the behavior of some operations in objects (like

List) that build upon the Text object.

IGNORECASE, NOIGNORECASE - controls whether string

comparison is case sensitive or not.

URLENCODED - If set then all Text operations are

carried out after translating all %HH or %uHHHH encodings in the

underlying byte stream.

1. %HH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http%3a//" will be decoded to

   "http://"
2. %uWWHH will be translated to the character sequence YZ where

   WW denotes the ascii value of character Y in hexadecimal and HH

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"

NOURLENCODED - Indicates that no percent encodings are present in

the Text object. No percent decoding will be performed on the Text

object.

PLUS_AS_SPACE - If set then all Text operations are carried out

after converting occurrences of the plus character to whitespace.

For example, the text "hello+world" will be decoded to "hello world"

NO_PLUS_AS_SPACE - If set, then the plus character will not be

converted to whitespace in the Text object.

BACKSLASH_ENCODED - If set, then all Text operations are carried

out after performing the following operations:

1. \XXX will be translated to the character Y where XXX denotes the

   ascii value of character Y in octal. The valid range of octal

   values for this type of encoding is \0 to \377.

   For example, the encoded text "http\72//" and "http\072//" will

   both be decoded to "http://"
2. \xHH will be translated to the character Y where HH denotes the

   ascii value of character Y in hexadecimal.

   For example, the encoded text "http\x3a//" will be decoded to

   "http://"
3. \\uWWXX will be translated to the character sequence YZ where WW

   denotes the ascii value of character Y in hexadecimal and XX

   denotes the ascii value of character Z in hexadecimal.

   For example, the encoded text "http%u3a2f/" and "http%u003a//"

   will both be decoded to"http://"
4. Occurences of "\b", "\n", "\t", "\f" and "\r" are translated to

   the corresponding ascii characters.

NO_BACKSLASH_ENCODED - If set, then backslash decoding will not be

performed on the Text object.

BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF is raised if either the

URLENCODED or BACKSLASH_ENCODED mode is set and a bad encoding

corresponding to the specified encoding mode is encountered.

NO_BAD_ENCODE_RAISE_UNDEF - If set, then UNDEF will not be raised

when a bad encoding is encountered in the Text object.

The Text operations operate on characters (as opposed to operating

on bytes). By default a single byte represents a character

using the ASCII encoding. But if URLENCODED mode is set then

3 bytes may represent a character. A PREFIX(3) operation

will select the first 3 characters in the Text object. That

means it may select up to 9 bytes for URLENCODED Text

object.

In the case of multibyte decodings as a result of %uXXXX or \\uXXXX,

Text operations which attempt to treat the multiple decoded bytes as

distinct characters are not allowed.

Parameters(expressions not allowed):

mode -

```
Text Mode</p></div><p><b>Returns: </b><a
href="vendor_spec_radius_avp_t.html">vendor_spec_radius_avp_t</a></p></
div></div></div><div><div><p><b>* </b><b>SKIP</b>(char c, uint i)</
p><div><p>Select the Text after skipping over a prefix of the current
Text.
```

The skipped over prefix is the longest prefix that has at most

Count (2nd argument) occurrences of character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SKIP</b>(uint i)</p><div><p>Select the Text after skipping over a
prefix in the
```

current Text. The length of the prefix is equal to the

number in the argument. If there aren't enough characters in

the Text object then the entire Text is skipped over.

Parameters(expressions not allowed):

i -

```
Prefix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>STARTSWITH_ANY</b>(<a
href="text_t.html">text_t</a>patset_name)</p><div><p>Results in Boolean
TRUE value if the Text starts with any of
```

the strings that are bound to the patset or dataset named by the

string argument.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH_INDEX</b>(<a href="text_t.html">text_t</a>patset_name)</
p><div><p>Returns the numerical index of the matching string, if the Text
```

starts with any of the strings that are bound to the patset

or dataset named by the string argument. Returns 0 if there

is no match.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="num_at.html">num_at</a></p></div></div></div><div><div><p><b>* </
b><b>STARTSWITH</b>(<a href="text_t.html">text_t</a>start)</
p><div><p>Results in Boolean TRUE value if a prefix of the  Text
```

object matches the string argument.

Parameters:

start -

```
String</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</a></
p></div></div></div><div><div><p><b>* </b><b>STRIP_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>From the selected text remove
the characters specified
```

in the string.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_CHARS('.=') or STRIP_CHARS('..=') the

selected text will become /fooxyzbar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_CHARS</b>(<a
href="text_t.html">text_t</a>s)</p><div><p>Selects the Text after
stripping off characters in the
```

end of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_END_CHARS('ar') the

selected text will become /foo.xyz=b /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_END_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

end of the current Text object.

Returns: text_t

*** STRIP_START_CHARS**(text_ts)

Selects the Text after stripping off characters in the

beginning of the current Text object.

For example: Say the selected text is / "foo.xyz=bar" /.

Then on executing STRIP_START_CHARS('of') the

selected text will become /.xyz=bar /. Note that repeating

character has same effect as it would be if it present only

once.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>STRIP_START_WS</b></
p><div><p>Selects the Text after stripping off white space in the
```

beginning of the current Text object.

Returns: text_t

*** SUBSTR_ANY**(text_tpatset_name)

Select any one sub-string that matches any string

in the given patset or dataset.

The patset cannot have strings longer than 255 characters.

Parameters(expressions not allowed):

patset_name -

```
Pattern Set Name</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(uint start, uint len)</p><div><p>Select the Text object of
the given length at the given
```

offset. Offset it 0 based. If there aren't enough characters

after the offset then all the remaining characters are

selected.

Parameters:

start -

```
                        Offset</p><p>len -



                        Length</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUBSTR</b>(<a href="text_t.html">text_t</a>s)</p><div><p>Select the
first Text object that matches the input string
```

argument.

The input string cannot be bigger than 255 characters.

Parameters(expressions not allowed):

s -

```
String</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></p></
div></div></div><div><div><p><b>* </b><b>SUFFIX</b>(char c, uint i)</
p><div><p>Select a suffix of the Text. The selected suffix is the
```

longest possible suffix containing at most Count (2nd arg)

occurrences of the character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>SUFFIX</b>(uint i)</p><div><p>Select a suffix of the Text. The length
of the
```

suffix is equal to the number in the argument. If there

aren't enough characters in the Text object then the entire

Text is selected.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TO_LOWER</b></
p><div><p>Converts the selected text to lower case.<br /><br /><br /><br /
>For example, the string "ABCd" is converted to "abcd"</p><p><b>Returns:
```

```
</b><a href="text_t.html">text_t</a></p></div></div></
div><div><div><p><b>* </b><b>TO_UPPER</b></p><div><p>Converts the selected
text to upper case.<br /><br /><br /><br />For example, the string "abcD"
is converted to "ABCD"</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(char c, uint i)</p><div><p>Select the Text after
truncating a suffix from the current
```

Text object. The truncated suffix is the longest possible

suffix containing at most Count (2nd arg) occurrences of the

character in the 1st argument.

Parameters:

c -

```
                        Character</p><p>i -



                        Count</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TRUNCATE</b>(uint i)</p><div><p>Select the Text after truncating a
suffix of the Text. The
```

length of the suffix truncated is equal to the number in the

argument. If there aren't enough characters in the Text then

then all the characters are truncated.

Parameters:

i -

```
Suffix Length</p></div><p><b>Returns: </b><a href="text_t.html">text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_AVP_T</b></
p><div><p>Convert text_t to avp_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(avp_t : Operations that can be performed on the returned AVPs.
```

Note: An AVP can have multiple instances in a Diameter message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(0).
- To get the fourth value of AVP code 266 from a request, use: DIAMETER.REQ.AVP(266).VALUE(3).
- To check if AVP code 1000 in the response has the vendor-specific bit set, use: DIAMETER.RES.AVP(1000).IS_VENDOR_SPECIFIC.

Returns: avp_t

*** TYPECAST_AVP_UNUM_T**

Convert text_t to avp_unum_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (avp_unum_t : In Diameter AVP based expressions can be written as DIAMETER.REQ.AVP(<avpcode>).
For example Auth-Application-Id AVP (AVP code 258) can be extracted as DIAMETER.REQ.AVP(258)

For few of the important AVPs we provide aliases.

For example this AVP based expression can also be written as

DIAMETER.REQ.AUTH_APPLICATION_ID.

This expression returns either a object of class avp_t or

returns a numerical value (object of class unsigned_long_at).

The numerical value for this object is derived from the value in first instance of

the AVP.)

Returns: avp_unum_t
*** TYPECAST_BOOL_AT**

Convert text_t to bool_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (bool_at : Boolean Operations)

Returns: bool_at

*** TYPECAST_COOKIE_T**

Convert text_t to cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (cookie_t : The HTTP Cookie type provides operations for a

cookie as it appears in the Set-Cookie and Set-Cookie2 header.

In this type the Name-Value List operations are also available. The

first delimiter is = and the element delimiter is ;. Multiple

consecutive ; are treated as one. (Same as applying the method

IGNORE_EMPTY_ELEMENTS on a Name-Value List. You can use the

Name-value List operations to access cookie attributes.

In this type Text operations are also available.

Each cookie begins with a COOKIE_NAME=COOKIE_VALUE pair followed by

zero or more semi-colon separated attribute-value pairs. If an

attribute appears more than once in a cookie, the value associated

with the first appearance of the attribute is returned.

Format Cookie:

Cookie1=Value;Version=decimal;Value;Domain=Value;Path=Value)

Returns: cookie_t

*** TYPECAST_DIAMETER_FLAG_T**

Convert text_t to diameter_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (diameter_flag_t : Provides operation to treat the underlying 1 byte as set of bit

flags. then set/unset/test/reset_all operations can be done on flag)

Returns: diameter_flag_t

*** TYPECAST_DNS_DOMAINNAME_T**

Convert text_t to dns_domainname_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_domainname_t : This object provides operations for parsing the DNS domain name as it

appears at various places in the DNS data. The format is

abc.foo.com.)

Returns: dns_domainname_t
*** TYPECAST_DNS_FLAG_T**

Convert text_t to dns_flag_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (dns_flag_t : Provides operations to test/set/unset a flag in dns header.)

Returns: dns_flag_t
*** TYPECAST_DOUBLE_AT**

Convert text_t to double_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_at : Double Value)

Returns: double_at
*** TYPECAST_DOUBLE_T**(double_format_eformat)

Convert text_t to double_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (double_t : Treat the underlying text as a Double in the given format. The

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="double_t.html">double_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="double_at.html">double_at</a>default_value)</p><div><p>Convert
text_t to double_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (double_t :
Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
double_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (double_t : Treat the
underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
```

```
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_DOUBLE_T</b>(<a
href="double_format_e.html">double_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to double_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(double_t : Treat the underlying text as a Double in the given format. The
```

supported formats are DECIMAL, DECIMAL_PREFIX. All

the formats strip off the beginning white space. DECIMAL

also strips off trailing white space. DECIMAL requires that all

the remaining characters must be a proper double value.

DECIMAL_PREFIX requires that at least one or more

more characters in the prefix must be a proper double value. In

case of overflow, there will be a parsing error.

In case of parsing error the default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Double</p></div><p><b>Returns: </b><a
href="double_t.html">double_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_HTTP_CACHE_CONTROL_T</b></p><div><p>Convert text_t to
http_cache_control_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type.
(http_cache_control_t : This object provides for the ability to work on
HTTP Cache-Control headers.)</p><p><b>Returns: </b><a
href="http_cache_control_t.html">http_cache_control_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_HTTP_CONTENT_LENGTH_T</b></
p><div><p>Convert text_t to http_content_length_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
```

```
destination type. (http_content_length_t : The HTTP.REQ.CONTENT_LENGTH or
HTTP.RES.CONTENT_LENGTH object can be
```

treated as a HTTP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: http_content_length_t

*** TYPECAST_HTTP_COOKIE_T**

Convert text_t to http_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_cookie_t : This object provides operations for parsing the HTTP cookie header.

The name-value pair is derived from the value in the last instance of

Cookie header. The format is Cookie : name0=value0; name1=value1;

An HTTP Cookie header can have multiple occurrences. COUNT will

return a Number containing how many headers of the current type

exist.

The system does not keep track of more than 15 HTTP headers. The

result will be undefined if there are more than 15 instances of

the header.)

Returns: http_cookie_t

*** TYPECAST_HTTP_DATE_T**

Convert text_t to http_date_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_date_t : The HTTP.REQ.DATE or HTTP.RES.DATE object can be treated as a HTTP

header or a Time object. The Time value is derived from the value in

the last instance of the Date header. The date formats supported are

RFC 822 (updated by RFC 1123), RFC 1036 (obsoletes RFC850) and ANSI C

asctime() formats.)

Returns: http_date_t

*** TYPECAST_HTTP_HEADER_T**(text_tname)

Convert text_t to http_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_header_t : When typecasting a Text object to HTTP Header a header name

must be provided.For example:

HTTP.REQ.HEADER("MyHdr").TYPECAST_HTTP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="http_header_t.html">http_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_HTTP_HOSTNAME_T</b></
p><div><p>Convert text_t to http_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (http_hostname_t : This object provides operations for
parsing the HTTP hostname as it
```

appears at various places in the HTTP protocol data. The format is

abc.foo.com:8080.)

Returns: http_hostname_t

*** TYPECAST_HTTP_METHOD_T**

Convert text_t to http_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_method_t : This object provides Text operations and HTTP Request Method Enum

operations on the HTTP request method.)

Returns: http_method_t

*** TYPECAST_HTTP_SET_COOKIE_T**

Convert text_t to http_set_cookie_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_set_cookie_t : HTTP.RES.SET_COOKIE provides operations for parsing and operating on the

SET-COOKIE header that is present in the HTTP response.

HTTP Set-Cookie header supports Text and HTTP Header operations.

For most Text operations the value in the last Set-Cookie header is

selected.

HTTP Set-Cookie header also supports special operations to extract

cookies from the response. It is assumed that there will be only one

cookie in a Set-Cookie header.)

Returns: http_set_cookie_t
*** TYPECAST_HTTP_URL_T**

Convert text_t to http_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_url_t : HTTP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the HTTP request header.

The supported format is <path>?<query>. The

Text object backing a HTTP URL object has the Text Mode set to

URLENCODED by default.)

Returns: http_url_t
*** TYPECAST_HTTP_VERSION_T**

Convert text_t to http_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (http_version_t : HTTP.REQ.VERSION or HTTP.RES.VERSION provide operations to extract HTTP

version information.)

Returns: http_version_t
*** TYPECAST_IPV6_ADDRESS_AT**

Convert text_t to ipv6_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_address_at : Provides operations for the 128-bit IPv6 address value.

Some general information regarding IPv6 expressions:

IPv6 PIXL expressions can be combined with other PIXL expressions

```
using the '+' operator.  The output is expected to be the

concatenation of the string representations of the individual PIXL

expressions.  No other operator other than '+' is expected to work

when any of the operands is an IPv6 PIXL expression.  Also, IPv6

PIXL expressions can be combined with any other type of PIXL

expression except a PIXL expression that produces a bool result.

In other words, if A and/or B are two IPv6 addresses, A+B should result

in the string AB.  When one of A or B is an IPv6 address, the other

can only be a number or a string and not a boolean.  Further, when
```

```
A and/or B is an IPv6 address, the only operator allowed between



them is '+'.



<br /><br /><br /><br />Example: Let the client source IPv6 address be
ABCD:1234::ABCD.



Let the server destination IPv4 address be 10.100.10.100.



Then the expression CLIENT.IPV6.SRC + SERVER.IP.DST should



produce the output "ABCD:1234::ABCD10.100.10.100".



<br /><br />Similarly, CLIENT.IPV6.SRC + 5 should produce the result



ABCD:1234::ABCD5)</p><p><b>Returns: </b><a
href="ipv6_address_at.html">ipv6_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_IPV6_ADDRESS_T</b></
p><div><p>Convert text_t to ipv6_address_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (ipv6_address_t : This object represents an IPv6 address
present in the
```

1000:1001:CD10:0000:0000:89AB:4567:CDEF format.)

Returns: ipv6_address_t

*** TYPECAST_IPV6_SUBNET_AT**

Convert text_t to ipv6_subnet_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_at : Provides operations for the integral IPv6 subnet address value.)

Returns: ipv6_subnet_at
*** TYPECAST_IPV6_SUBNET_T**

Convert text_t to ipv6_subnet_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ipv6_subnet_t : This object represents an IPv6 subnet address present in the

1000:1001:CD10:0000:0000:89AB:4567:CDEF/64 format.)

Returns: ipv6_subnet_t
*** TYPECAST_IP_ADDRESS_AT**

Convert text_t to ip_address_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_at : Provides operations for the 32-bit integral IP address value.)

Returns: ip_address_at
*** TYPECAST_IP_ADDRESS_T**

Convert text_t to ip_address_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ip_address_t : Represents an IP address present in the 10.100.200.1

format.)

Returns: ip_address_t
*** TYPECAST_LIST_T**(char sep)

Convert text_t to list_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (list_t : The character separator has to be provided when typecasting

a textual context to list type. The current text mode has no

effect on the character separator. For example if the

current text mode is IGNORECASE and the separator is p then

P will not be treated as a separator.)

Parameters(expressions not allowed):

sep -

```
Separator</p></div><p><b>Returns: </b><a href="list_t.html">list_t</a></
p></div></div></div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_AT</b></
p><div><p>Convert text_t to mac_address_at. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (mac_address_at : Provides operations for the 48-bit MAC
address value.)</p><p><b>Returns: </b><a
href="mac_address_at.html">mac_address_at</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_MAC_ADDRESS_T</b></p><div><p>Convert
text_t to mac_address_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type.
(mac_address_t : Represents a MAC address present in the 12:34:56:78:9a:bc
```

format.)

Returns: mac_address_t
*** TYPECAST_NUM_AT**

Convert text_t to num_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_at : Number Value)

Returns: num_at
*** TYPECAST_NUM_T**(num_format_eformat)

Convert text_t to num_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (num_t : Treat the underlying text as a Number in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a href="num_t.html">num_t</a></p></
div></div></div><div><div><p><b>* </b><b>TYPECAST_NUM_T</b>(<a
href="num_format_e.html">num_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
num_t. Will raise an undef if the value is null, empty, or the value has
an invalid format for the destination type. (num_t : Treat the underlying
text as a Number in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="num_t.html">num_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2)</p><div><p>Convert text_t
to nvlist_t. Will raise an undef if the value is null, empty, or the value
has an invalid format for the destination type. (nvlist_t : Two character
delimiters have to be provided when
```

typecasting a Text object to a Name-Value List.

The first delimited separates the name from the value.

The second delimiter separates list items.

For example typecast_nvlist_t('=', '&') handles a URL

query string.

The Text Mode has no effect on the character delimiters.

For example if the current text mode is IGNORECASE and a

delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></div><div><div><p><b>*
</b><b>TYPECAST_NVLIST_T</b>(char sep1, char sep2, char quote)</
p><div><p>Convert text_t to nvlist_t. Will raise an undef if the value is
null, empty, or the value has an invalid format for the destination type.
(nvlist_t : Two character delimiters and quote-character have to be
provided
```

when typecasting a Text object to a Name-Value List that supports

quote-character.

The first delimited separates the name from the value.

The second delimiter separates list items.

The third value specifies the quote character.

For example typecast_nvlist_t('=', ';', '"') handles a

name-value list separated by semi-colons that allows quoting.

The quote-character specifies that the value can occur within the

quotes and the delimiters within the quote-characters should not

be taken into account.

The Text Mode has no effect on the character

delimiters. For example if the current text mode is IGNORECASE and

a delimiter is p then P will not be treated as a delimiter.)

Parameters(expressions not allowed):

sep1 -

```
                        delimiter1</p><p>sep2 -



                        delimiter2</p><p>quote -



                        quote-character.</p></div><p><b>Returns: </b><a
href="nvlist_t.html">nvlist_t</a></p></div></div></
div><div><div><p><b>TYPECAST_RADIUS_AVP_T</b></p><div><p>Convert
vendor_spec_radius_avp_t to radius_avp_t. Will raise an undef if the value
is null, empty, or the value has an invalid format for the destination
type. (radius_avp_t : Operations that can be performed on the returned
AVPs.
```

Note: An AVP can have multiple instances in a radius message.

However, the system can access only the first 16 instances.

Example:

- To get the first value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(0).
- To get the fourth value of AVP code 26 from a request, use: radius.REQ.AVP(26).VALUE(3).

Returns: radius_avp_t

*** TYPECAST_RADIUS_AVP_VALUE_T**

Convert text_t to radius_avp_value_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (radius_avp_value_t : Provides operations to typecast RADIUS AVP value to
Integer value or IP Address.)

Returns: radius_avp_value_t
*** TYPECAST_SEQ_TEXT_T**

Convert text_t to seq_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (seq_text_t : Represents the Text Sequence class.)

Returns: seq_text_t
*** TYPECAST_SERVICEPATH_T**

Convert text_t to servicepath_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (servicepath_t : Operations that can be performed on the service path.

Example:

- To get the name of the service path applicable to current subscriber, use: SUBSCRIBER.SERVICEPATH
- To determine If the service function is the next in the service path applicable to scubscriber, use: SUBSCRIBER.SERVICEPATH.IS_NEXT(<service>).

Returns: servicepath_t

*** TYPECAST_SIP_CALLID_HEADER_T**

Convert text_t to sip_callid_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_callid_header_t : This object provides operations for parsing the SIP callid header.)

Returns: sip_callid_header_t
*** TYPECAST_SIP_CONTENT_LENGTH_T**

Convert text_t to sip_content_length_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_content_length_t : The SIP.REQ.CONTENT_LENGTH or SIP.RES.CONTENT_LENGTH object can be
treated as a SIP Header or as a Number. The numerical value for

this object is derived from the value in the last instance of the

Content-Length header.)

Returns: sip_content_length_t

*** TYPECAST_SIP_FROM_HEADER_T**

Convert text_t to sip_from_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_from_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_from_header_t

*** TYPECAST_SIP_HEADER_T**(text_tname)

Convert text_t to sip_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_header_t : When typecasting a Text object to SIP Header a header name

must be provided.For example:

SIP.REQ.HEADER("MyHdr").TYPECAST_SIP_HEADER_T("InHdr"))

Parameters(expressions not allowed):

name -

```
Header Name</p></div><p><b>Returns: </b><a
href="sip_header_t.html">sip_header_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_SIP_HOSTNAME_T</b></
p><div><p>Convert text_t to sip_hostname_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (sip_hostname_t : This object provides operations for
parsing the SIP hostname as it
```

appears at various places in the SIP protocol data. The format is

abc.foo.com:5060.)

Returns: sip_hostname_t

*** TYPECAST_SIP_METHOD_T**

Convert text_t to sip_method_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_method_t : This object provides Text operations and SIP Request Method Enum

operations on the SIP request method.)

Returns: sip_method_t

*** TYPECAST_SIP_TO_HEADER_T**

Convert text_t to sip_to_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_to_header_t : This object provides operations for parsing the SIP From header.)

Returns: sip_to_header_t

*** TYPECAST_SIP_URL_T**

Convert text_t to sip_url_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_url_t : SIP.REQ.URL provides operations for parsing and operating on the

URL that is present in the first line of the SIP request header.

The supported format is <path>?<query>. The

Text object backing a SIP URL object has the Text Mode set to

URLENCODED by default.)

Returns: sip_url_t

*** TYPECAST_SIP_VERSION_T**

Convert text_t to sip_version_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_version_t : SIP.REQ.VERSION or SIP.RES.VERSION provide operations to extract SIP

version information.)

Returns: sip_version_t

*** TYPECAST_SIP_VIA_HEADER_T**

Convert text_t to sip_via_header_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sip_via_header_t : This object provides operations for parsing the SIP Via header.)

Returns: sip_via_header_t

*** TYPECAST_SSLVPN_MODE_T**

Convert text_t to sslvpn_mode_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (sslvpn_mode_t : This object provides Text operations and SSLVPN Mode Enum

operations on the SSLVPN request method.)

Returns: sslvpn_mode_t

*** TYPECAST_SSL_TEXT_T**(text_tname)

Convert text_t to ssl_text_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (ssl_text_t : Treat the underlying text object as SSL Text.)

Parameters(expressions not allowed):

name -

```
Name</p></div><p><b>Returns: </b><a href="ssl_text_t.html">ssl_text_t</
a></p></div></div></div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_T</
b></p><div><p>Convert text_t to tcp_options_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_t : TCP Protocol Options)</p><p><b>Returns:
</b><a href="tcp_options_t.html">tcp_options_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_OPTIONS_TYPE_T</b></
p><div><p>Convert text_t to tcp_options_type_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (tcp_options_type_t : TCP Protocol Options Type)</
p><p><b>Returns: </b><a
href="tcp_options_type_t.html">tcp_options_type_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_TCP_REPEATER_OPTION_T</b></
p><div><p>Convert text_t to tcp_repeater_option_t. Will raise an undef if
the value is null, empty, or the value has an invalid format for the
destination type. (tcp_repeater_option_t : TCP Protocol Option :
Repeater)</p><p><b>Returns: </b><a
href="tcp_repeater_option_t.html">tcp_repeater_option_t</a></p></div></
div></div><div><div><p><b>* </b><b>TYPECAST_TEXT_T</b></p><div><p>Convert
radius_avp_t to text_t. Will raise an undef if the value is null, empty,
or the value has an invalid format for the destination type. (text_t :
Designates any piece of text.)</p><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>TYPECAST_TIME_AT</b></p><div><p>Convert text_t to time_at. Will raise
an undef if the value is null, empty, or the value has an invalid format
```

```
for the destination type. (time_at : Treats a number value as number of
seconds since GMT Jan 1
```

1970 0h 0m 0s.)

Returns: time_at
*** TYPECAST_TIME_T**

Convert text_t to time_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (time_t : This object treats a Text object as a date string. The formats

supported are

1\. RFC822 :: Sun, 06 Nov 1994 08:49:37 GMT

2\. RFC850 :: Sunday, 06-Nov-94 08:49:37 GMT

3\. ASCTIME :: Sun Nov 6 08:49:37 1994

4\. ASCTIME like :: Sun Nov 6 08:49:37 1994

5\. ASCTIME like :: Sun Nov 06 08:49:37 1994

6\. Http Set-Cookie Expiry date :: Sun, 06-Nov-1994 08:49:37 GMT)

Returns: time_t
*** TYPECAST_UNSIGNED_LONG_AT**

Convert text_t to unsigned_long_at. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_at : Unsigned Long(64bit) Value)

Returns: unsigned_long_at
*** TYPECAST_UNSIGNED_LONG_T**(unsigned_long_format_eformat)

Convert text_t to unsigned_long_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (unsigned_long_t : Treat the underlying text as a Unsigned Long in the given format. The

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error.)

Parameters(expressions not allowed):

format -

```
Format</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="ip_address_at.html">ip_address_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="unsigned_long_at.html">unsigned_long_at</a>default_value)</
p><div><p>Convert text_t to unsigned_long_t. Will raise an undef if the
value is null, empty, or the value has an invalid format for the
destination type. (unsigned_long_t : Treat the underlying text as a
Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="num_at.html">num_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
```

```
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNSIGNED_LONG_T</b>(<a
href="unsigned_long_format_e.html">unsigned_long_format_e</a>format, <a
href="time_at.html">time_at</a>default_value)</p><div><p>Convert text_t to
unsigned_long_t. Will raise an undef if the value is null, empty, or the
value has an invalid format for the destination type. (unsigned_long_t :
Treat the underlying text as a Unsigned Long in the given format. The
```

supported formats are DECIMAL, HEX, DECIMAL_PREFIX, HEX_PREFIX. All

the formats strip off the beginning white space. DECIMAL and HEX

also strip off trailing white space. DECIMAL and HEX require that all

the remaining characters must be proper digits.

DECIMAL_PREFIX and HEX_PREFIX require that at least one or more

more characters in the prefix must be proper digits. In case of

overflow, there will be a parsing error. In case of parsing error the

default value will be assumed.)

Parameters(expressions not allowed):

format -

```
                        Format</p><p>default_value -



                        Number</p></div><p><b>Returns: </b><a
href="unsigned_long_t.html">unsigned_long_t</a></p></div></div></
div><div><div><p><b>* </b><b>TYPECAST_UNUM_TEXT_T</b></p><div><p>Convert
text_t to unum_text_t. Will raise an undef if the value is null, empty, or
the value has an invalid format for the destination type. (unum_text_t :
Provides operations to treat the underlying OctectString as Binary
```

String or unsigned long value .)

Returns: unum_text_t
*** TYPECAST_URL_CATEGORY_T**

Convert text_t to url_category_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (url_category_t : Retrieves URL category and attributes from an URL)

Returns: url_category_t
*** TYPECAST_VENDOR_SPEC_APPID_AVP_T**

Convert text_t to vendor_spec_appid_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_appid_avp_t : Retrieves AVPs that are nested in the grouped AVP

Vendor-Specific-Application-Id (code 260). The nested AVPs are:

- Vendor-Id
- Auth-Application-Id
- Acct-Application-Id

Vendor-Specific-Application-Id AVP:

- For requests:
  DIAMETER.REQ.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID
- For responses:
  DIAMETER.RES.VENDOR_SPECIFIC_APPLICATION_ID.AUTH_APPLICATION_ID

Returns: vendor_spec_appid_avp_t

*** TYPECAST_VENDOR_SPEC_RADIUS_AVP_T**

Convert text_t to vendor_spec_radius_avp_t. Will raise an undef if the value is null, empty, or the value has an invalid format for the destination type. (vendor_spec_radius_avp_t :

- Vendor-Id

To retrieve the Vendor-Id from the
Vendor-Specific AVP:

- RADIUS.REQ.VENDOR_SPECIFIC.VENDORID

Returns: vendor_spec_radius_avp_t

*** UNQUOTE**(char c)

If selected text is quoted by character c, Remove character c from
beginning and end.

For example: Say the selected text is / "foo xyz bar " /.

Then on executing UNQUOTE("") the selected text will become

/foo xyz bar /

Parameters(expressions not allowed):

c -

```
Character</p></div><p><b>Returns: </b><a href="text_t.html">text_t</a></
p></div></div></div><div><div><p><b>* </b><b>URLSET_MATCHES_ANY</b>(<a
href="text_t.html">text_t</a>urlset_name)</p><div><p>Results in Boolean
TRUE value if and only if the Text matches any of
```

the patterns that are specified in the urlset named by the

string argument. The matching is case-insensitive for

domains, but case-sensitive for paths.

For example:

- HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URLSET_MATCHES_ANY("urlset_forbidden") returns true when the combined domain and URL matches any pattern in the urlset whose name is "urlset_forbidden".

Parameters(expressions not allowed):

urlset_name -

```
urlset name</p></div><p><b>Returns: </b><a href="bool_at.html">bool_at</
a></p></div></div></div><div><div><p><b>* </b><b>URL_CATEGORIZE</b>(uint
min_reputation, uint max_reputation)</p><div><p>Returns the URL category
for the supplied URL with the given minimum
```

and maximum reputation score. Where a score of 5 is most risky and 1

is least risky. If either score is set to 0, that bound would be ignored.

By specifying the reputation scores, the function will returned the category

with the highest score (most risky) based on the bounds.

If there was an error getting the category or if the URL is invalid then it

will result in an Undef error.

For example:

HTTP.REQ.HOST.APPEND(HTTP.REQ.URL).URL_CATEGORIZE(0, 0)

returns value when the combined domain and URL is looked up for

the URL category with the highest reputation score (most risky).

Parameters(expressions not allowed):

min_reputation -

```
                        Minimum reputation score of URL category</
p><p>max_reputation -



                        Maximum reputation score of URL category</p></
div><p><b>Returns: </b><a href="url_category_t.html">url_category_t</a></
p></div></div></div><div><div><p><b>* </b><b>URL_RESERVED_CHARS_SAFE</b></
p><div><p>Converts URL reserved characters to corresponding '%XX'
```


values and retains the rest. This is a read only operation.

Based on RFC3986, following are considered URL unreserved characters:

Alpha-numeric characters: a-z, A-Z, 0-9

Hyphen and underscore: "-", "_"

Dot: "."

Tilde: "~"

representation of the input character.

If the requirement is to encode all but URL safe characters,

please consider using HTTP_URL_SAFE method.

Returns: text_t

*** VALUE**

Returns the value of first instance of the AVP as OctetString..

If there are multiple occurence of AVP(28).

To get the first instance the expression would be: radius.REQ.AVP(28).VALUE.

Returns: radius_avp_value_t

*** VALUE**(uint instanceno)

Returns value of an instance of the AVP.
The first instance of the AVP can be retrieved by simply using the

expression: radius.REQ.AVP(266). However, subsequent instances must be

retrieved using this method.

VALUE(0) retrieves the first instance, VALUE(1) retrieves the second

instance, and so on.

Parameters(expressions not allowed):

instanceno -

```
AVP instance number. Maximum value: 15.</p></div><p><b>Returns: </b><a
href="radius_avp_value_t.html">radius_avp_value_t</a></p></div></
div><div><div><p><b>VENDORID</b></p><div><p>Returns the Vendor-Id inside
the
```

Vendor_Specific AVP (AVP code 26).

Note:expression can be used to retrieve the value as Unsigned32

Returns: num_at
*** XML_DECRYPT**(text_tcertKeyName)

Decrypt the selected content, which was encrypted using the W3C XML

Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing the RSA
```

private key to be used to decrypt the bulk encryption key in the

selected EncryptedData element.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

Returns: text_t

*** XML_ENCRYPT**(text_tcertKeyName, encrypt_method_emethod, uint flags)

Encrypt the selected cleartext using the W3C XML Encryption standard.

Parameters(expressions not allowed):

certKeyName -

```
identifies the certkey object containing an X.509
```

certificate with the RSA public key to be used to encrypt the

generated bulk encryption key. Note that the certKey object need not,

and usually will not, include the associated RSA private key.

method -

```
specifies the bulk encryption method (rc4, des3, aes128,
```

aes192, or aes256) used to encrypt the content.

flags -

```
optional bit mask to specify optional elements in the
```

EncryptedData element. These are hints to tell the receiver what private

key to use to decrypt the bulk encryption key, providing information from

the certKey's X.509 certificate.

- 1 - include a KeyName element with the certKeyName;
- 2 - include a KeyValue element with the RSA public key from the certificate;
- 4 - include an X509IssuerSerial element with the certificate serial number and issuer DN;
- 8 - include an X509SubjectName element with the certificate subject DN;
- 16 - include an X509Certificate element with the entire certificate;

Returns: text_t

*** XPATH_HTML_WITH_MARKUP**(xpath_str xpathex)

Applies an XPath expression to the DOM for an HTML document
and returns the selected node-set or string, including

markup like element tags. node-sets are converted to

strings using the standard XPath string conversion routine.

The selected portion of the document can be used as the

target for rewrite actions.

Examples:

```
HTTP.REQ.BODY(1000).XPATH_HTML_WITH_MARKUP(xp%//html/head/title%)
```

```
  returns the &lt;title&gt; element in the document's



  &lt;head&gt; This can be used in a rewrite action



  to replace or delete the &lt;title&gt; element.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_bool xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a boolean value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%boolean(/html/head/meta)%)



  returns true if the &lt;meta&gt; element exists as a child



  of &lt;head&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                       XPath Expression</p></div><p><b>Returns: </b><a
href="bool_at.html">bool_at</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_HTML</b>(xpath_double xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and a double value.

Examples:

```
  HTTP.RES.BODY(1000).XPATH_HTML(xp%number(//video/@height)%)



  returns the numeric value of the height attribute of the



  first &lt;video&gt; element.</p><p><b>Parameters</b>(expressions not
allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="double_at.html">double_at</a></p></div></div></div><div><div><p><b>*
</b><b>XPATH_HTML</b>(xpath_str xpathex)</p><div><p>Applies an XPath
expression to the DOM for an HTML document
```

and returns the selected node-set or string. node-sets are

converted to strings using the standard XPath string

conversion routine.

Examples:

```
  HTTP.REQ.BODY(1000).XPATH_HTML(xp%//img/@src%)



  returns the string value of the src attribute of



  the first &lt;img&gt; element in the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
```

```
b><b>XPATH_JSON_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath_json_with_markup operates on
```

XML file and returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },
```

```
      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON_WITH_MARKUP(xp%/Top/Book/creator%)

    This expression when operated on the above JSON file,

    will select "all the nodes enclosed by

    /Book/creator"

    in the first 1000 bytes of the body which is

      creator:{

         person:{

           name:'Milton'
```

```
            }


          }


      The original JSON document is not disturbed but the


      part of the JSON body selected by the


      expression is marked for further processing.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_JSON</b>(xpath_bool xpathex)</p><div><p>xpath(1.0) expression
evaluation. xpath_json operates on JSON file
```

and returns boolean value.

Examples:

```
  For the JSON file:


  {


     "Book":{
```

```
      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }

}

HTTP.REQ.BODY(1000).XPATH_JSON(xp%boolean(/Top/Book/creator)%)

    This will return true as there is a node named

    'creator' with parent 'Book' in the first 1000 bytes of the
```

```
      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns double value.

Examples:

```
  For the JSON file:



  {



     "Book":{



        "creator":{



           "person":{



             "name":'Milton'



           }
```

```
        },

        "title":'Paradise Lost',

        "price":"36"

     }

  }

  HTTP.REQ.BODY(1000).XPATH_JSON(xp%number(/Top/Book/price)%)

      This will convert the string '36' string to double if it is

      in the first 1000 bytes of the document.</p><p><b>Parameters</
b>(expressions not allowed):</p><div><p>xpathex -

                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH_JSON</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath_json operates on JSON
file
```

and returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
For the JSON file:

{

   "Book":{

      "creator":{

         "person":{

           "name":'Milton'

         }

      },

      "title":'Paradise Lost'

   }
```

```
}


HTTP.REQ.BODY(1000).XPATH_JSON(xp%/Top/Book%)


    This will select "all the nodes enclosed by /Book"


    in the first 1000 bytes of the body.


    Since this is a node-set the corresponding string


    value will be returned which is "MiltonParadise


    Lost".</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -


                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div><div><div><p><b>* </
b><b>XPATH_WITH_MARKUP</b>(xpath_str xpathex)</p><div><p>xpath(1.0)
expression evaluation. xpath operates on XML file and
```

returns a string that contains the entire portion of the

document for the result node including markup, such as

including the enclosing element tags

Examples:

<book><creator><person><name>Milton</name></person></creator><title>Paradise Lost</title></book>HTTP.REQ.BODY(1000).XPATH_WITH_MARKUP(xp%/Book/creator%)

This expression when operated on the above XML file,

will select "all the nodes enclosed by

/Book/creator"

in the first 1000 bytes of the body which is

<creator><person><name>Milton</name></person></creator>The original XML document is not disturbed but the

part of the XML body selected by the

xpath expression is marked for further processing. Parameters(expressions not allowed): xpathex - XPath Expression Returns: text_t * XPATH(xpath_bool xpathex) xpath(1.0) expression evaluation. xpath operates on XML file and

returns boolean value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%boolean(//Book/creator)%)



      This will return true as there is a node called creator



      under the node Book with in the first 1000 bytes of the



      document.</p><p><b>Parameters</b>(expressions not allowed):</
p><div><p>xpathex -



                        XPath Boolean Expression</p></div><p><b>Returns:
</b><a href="bool_at.html">bool_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_double xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns double value.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;price>36&lt;/
price>&lt;/book>HTTP.REQ.BODY(1000).XPATH(xp%number(/Book/price)%)
```

```
      This will convert the string '36' to double



      if it is in the first 1000 bytes of the document.</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Numeric Expression</p></div><p><b>Returns:
</b><a href="double_at.html">double_at</a></p></div></div></
div><div><div><p><b>* </b><b>XPATH</b>(xpath_str xpathex)</
p><div><p>xpath(1.0) expression evaluation. xpath operates on XML file and
```

returns node-set or string.

node-sets are converted into corresponding strings using the

standard xpath string conversion routine.

Examples:

```
  For the XML file:



  &lt;book>&lt;creator>&lt;person>&lt;name>Milton&lt;/name>&lt;/
person>&lt;/creator>&lt;title>Paradise Lost&lt;/title>&lt;/
book>HTTP.REQ.BODY(1000).XPATH(xp%/Book/creator%)



      This expression will select



      "all the nodes enclosed by /Book/creator"



      in the first 1000 bytes of the body. Since this is a node-set
```

```
      its string value is returned which is: "Milton"</
p><p><b>Parameters</b>(expressions not allowed):</p><div><p>xpathex -



                        XPath Expression</p></div><p><b>Returns: </b><a
href="text_t.html">text_t</a></p></div></div></div>&lt;/body>&lt;/html>
```

VLAN_REQ_T

VLAN Request

This topic lists the expressions that are provided by this class.

ID

Returns the ID of the VLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

VLAN_RES_T

VLAN Response

This topic lists the expressions that are provided by this class.

ID

Returns the ID of the VLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

VSERVER_STATE_E

This object represents state of the Vserver.

This topic lists the expressions that are provided by this class.

Enumerations

DOWN

State is "DOWN" for vserver that is DOWN (not running).

OUT_OF_SERVICE

State is "OUT_OF_SERVICE" for vserver that is out of service.

No monitoring packets and no requests are sent to the server

that is out of service.

UP

State is "UP" for vserver that is UP (running).

Expressions

EQ(vserver_state_em)

The result is Boolean TRUE if the input argument is equal to

the Vserver State represented by this object.

Parameters(expressions not allowed):

m -

```
Vserver State Name
```

Returns: bool_at

NE(vserver_state_em)

The result is Boolean TRUE if the input argument is not equal to

the Vserver State represented by this object.

Parameters(expressions not allowed):

m -

```
Vserver State Name
```

Returns: bool_at

VSERVER_T

Operates on a vserver

This topic lists the expressions that are provided by this class.

ACTIVESERVICES

Returns number of active services

Raises undef if used on non ADDR based vservers

Returns: num_at

ACTIVETRANSACTIONS

Returns current active transaction counter accumulated on vserver level

Returns: unsigned_long_at

AVERAGESURGECOUNT

Returns average number of requests in surge queue per active service

Raises undef if used on non ADDR based vservers

Returns: double_at

CONNECTIONS

Returns number of connections on the vserver as Unsigned

long number.

Returns: unsigned_long_at

DETERMINE_SERVICES(server_attr_eattr, text_tseparator)

Determines the services selected by the LoadBalancing Algorithm

on a given vserver and returns a list of them in the preference order.

Parameters(expressions not allowed):

attr -

```
- Service Attribute which could be service description - 'DESCRIPTION',
```

service name - 'NAME', service IP - 'IP', service port - 'PORT',

service IP:port combination - 'IPPORT'

separator -

```
- The string inserted between each of the returned services
```

Returns: text_t

HEALTH

Returns as number, the percentage of UP services bound to the

vserver.

Returns: num_at

IS_DYNAMIC_LIMIT_REACHED

Returns in Boolean TRUE if number of connections on a vserver reached the dynamic threshold,

computed as a sum of max clients on all active services under the vserver.

Returns: bool_at

MAXAUTOSCALEMEMBERS

Returns the configured maximum number of autoscale members

Raises undef if used on non ADDR based vservers

Returns: num_at

MINAUTOSCALEMEMBERS

Returns the configured minimum number of autoscale members

Raises undef if used on non ADDR based vservers

Returns: num_at

NAME

Returns the name of the vserver(CS/LB).

Returns: text_t

RESPTIME

Returns as number, the response time in milli-seconds.

RESPTIME is the average TTFB (Time To recieve First Byte) from

all the backend services bound to this vserver.

Returns: num_at

SNMP_TABLE(uint index)

Returns the load entry in the SNMP table at given index

Parameters(expressions not allowed):

index -

```
Index
```

Raises undef if used on non ADDR based vservers or

vserver does not have SNMP load table or

the index is out of boundary (valid index values are 0-9)

Returns: snmp_table_t

STATE

Returns the state of vserver.

Returns: vserver_state_e

SURGECOUNT

Returns as number, the number of requests in surge queue of vserver.

Returns: unsigned_long_at

THROUGHPUT

Returns throughput of the vserver in Mbps (Megabits per sec) as

Unsigned long number.

Returns: unsigned_long_at

VXLAN_REQ_T

VXLAN Request

This topic lists the expressions that are provided by this class.

ID

Returns the VNID of the VXLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

VXLAN_RES_T

VXLAN Response

This topic lists the expressions that are provided by this class.

ID

Returns the VNID of the VXLAN through which the current packet entered

the Application Switch as a number.

Returns: num_at

X509_CERTIFICATE_T

X.509 SSL Certificate

This topic lists the expressions that are provided by this class.

AUTH_KEYID

Returns a string containing the Authority Key Identifier extension

of the X509 V3 certificate.

Returns: auth_keyid_t

CERT_POLICY

Returns a string containing the client certificate policy.

Note that this represents a sequence of certificate policies.

Returns: seq_text_t

DAYS_TO_EXPIRE

Returns the number of days before which the certificate is valid.

Returns -1 for expired certificates.

Returns: num_at

EXISTS

Returns TRUE if the client has an SSL client certificate.

Returns: bool_at

ISSUER

Returns the Distinguished Name of the Issuer as a separated

Name-Value list using the delimiter '=' to separate name and value

and the delimiter '/' to separate name-value pairs.

Example:

/C=IN/ST=KA/L=BLR/O=Netscaler Pvt Ltd/OU=www.netscaler.com/CN=www.netscaler.com/emailAddress=ns@ns.co

Returns: nvlist_t

IS_VALID

Returns true if and only if the client certificate is valid.

Returns: bool_at

KEY_USAGE(key_usage_ek)

Returns a boolean value indicating whether

the specified key usage extension bit value in the X509

certificate is set.

Parameters(expressions not allowed):

k -

```
The argument specifies
```

which bit is checked. Valid arguments are:

DIGITAL_SIGNATURE. Returns TRUE if the

digital signature bit is set; FALSE otherwise.

NONREPUDIATION. Returns TRUE if the nonrepudiation

bit is set; FALSE otherwise.

KEYENCIPHERMENT. Returns TRUE if the key

encipherment bit is set; FALSE otherwise.

DATAENCIPHERMENT. Returns TRUE if the data

encipherment bit is set; FALSE otherwise.

KEYAGREEMENT. Returns TRUE if the key

agreement bit is set; FALSE otherwise.

KEYCERTSIGN. Returns TRUE if the key cert

sign bit is set; FALSE otherwise.

CRLSIGN. Returns TRUE if the CRL bit is set;

FALSE otherwise.

ENCIPHERONLY. Returns TRUE if the encipher

only bit is set; FALSE otherwise.

DECIPHERONLY. Returns TRUE if the decipher

only bit is set; FALSE otherwise.

Returns: bool_at

PK_ALGORITHM

Returns the name of the public key algorithm used by the SSL key.

Returns: text_t

PK_SIZE

Returns the size of the public key used in the certificate.

Returns: num_at

SERIALNUMBER

Returns the serial number of the client certificate.

In the case of non-SSL transaction or error in certificate, returns

an empty string.

Returns: text_t

SIGNATURE_ALGORITHM

Returns the name of the cryptographic algorithm used by the CA to sign

this certificate.

Returns: text_t

SUBJECT

Returns the Distinguished Name of the Issuer as a separated

Name-Value list using the delimiter '=' to separate name and value and

the delimiter '/' to separate name-value pairs.

Example:

/C=IN/ST=KA/L=BLR/O=Netscaler Pvt Ltd/OU=www.netscaler.com/CN=www.netscaler.com/emailAddress=ns@ns.co

Returns: nvlist_t

SUBJECT_KEYID

Returns the Subject KeyID of the client certificate, if one exists.

If there is no Subject KeyID, returns a zero-length text object.

Returns: text_t

TO_PEM

Returns the SSL certificate in binary format

Returns: text_t

VALID_NOT_AFTER

Returns the date after which the client certificate is no longer valid.

Returns: time_at

VALID_NOT_BEFORE

Returns the date before which the client certificate is not valid.

Returns: time_at

VERSION

Returns the version number of the certificate.

If the connection is not SSL-based, returns zero (0).

Returns: num_at