CHAPTER 3

SIP Messages APIs

The Lua scripting environment provides a set of APIs that allows messages to be manipulated. These APIs are explained under the following categories:

Manipulating the Request or Response line

getRequestLine

```
getRequestLine() returns the method, request-uri, and version
```

This method returns three values:

- The method name
- The request-uri, and
- The protocol version.

If this API is invoked for a response, it triggers a run-time error.

Example: Set the values of local variables for the method, request-uri, and the protocol version.

Script

```
M = {}

function M.outbound_INVITE(msg)
    local method, ruri, ver = msg:getRequestLine()
end

return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

Output/Result

```
Local variables method, ruri, and version are set accordingly:
method == "INVITE"
ruri == "sip:1234@10.10.10.1"
version == "SIP/2.0"
```

getRequestUriParameter

```
getRequestUriParameter(parameter-name) returns the URI parameter-value
```

Given the string name of a request URI parameter, this function returns the value of the specified URI parameter.

- If the URI parameter is a tag=value format, the value on the right of the equal sign is returned.
- If the URI parameter is a tag with no value, the value will be a blank string (i.e. "").
- If the URI parameter does not exist in the first header value, nil will be returned.
- If this method is invoked for a response message, nil will be returned.

Example: Set a local variable to the value of the user parameter in the request-uri.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local userparam = msg:getRequestUriParameter("user")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1;user=phone SIP/2.0
```

Output/Results

Local variable userparam is set to the string "phone"

setRequestUri

```
setRequestUri(uri)
```

This method sets the request-uri in the request line. If this API is invoked for a response, it triggers a run-time error.

Example: Set the request-uri to tel:1234.

Script

```
M = {}
function M.outbound_INVITE(msg)
    msg:setRequestUri("tel:1234")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

Output/Result

```
INVITE tel:1234 SIP/2.0
```

getResponseLine

```
getResponseLine() returns version, status-code, reason-phrase
```

This method returns three values: the protocol version (string), the status-code (integer), and the reason-phrase (string). If this API is invoked for a request, it triggers a run-time error.

Example: Set local variables to the values of the protocol version, the status code, and the reason phrase.

Script

```
M = {}
function M.outbound_200_INVITE(msg)
    local version, status, reason = msg:getResponseLine()
end
return M
```

Message

```
SIP/2.0 200 Ok
.
.
CSeq: 102 INVITE
```

Output/Result

```
Local variables method, ruri, and version are set accordingly:
version == "SIP/2.0"
status == 200
reason == "Ok"
```

setResponseCode

```
setResponseCode(status-code, reason-phrase)
```

This method sets the status-code and reason-phrase. Either value is allowed to be nil. If nil is specified for the value, the existing value stored in the message is used. If this API is invoked for a request, it triggers a run-time error.

Example: Change an outgoing 404 response to a 604 response.

Script

```
M = {}
function M.outbound_404_INVITE(msg)
    msg:setResponseCode(604, "Does Not Exist Anywhere")
end
return M
```

Message

```
SIP/2.0 404 Not Found
.
.
CSeq: 102 INVITE
```

Output/Results

```
SIP/2.0 604 Does Not Exist Anywhere
.
.
CSeq: 102 INVITE
```

Manipulating Headers

allowHeader

```
allowHeaders is a Lua table specified by the script writer
```

Cisco Unified CMs default behavior with respect to the headers that it does not recognize is to ignore such headers. In some cases it is desirable to use such a header during normalization or to pass the header transparently. By specifying the header name in the **allowHeaders** table, the script enables Cisco Unified CM to recognize the header and it is enough for the script to use it locally for normalization or pass it transparently.

Example:

The **allowHeaders** specifies the `History-Info`. Without it, the incoming History-Info headers gets dropped before script processing. Thus the **convertHIToDiversion** does not produce any useful results. This script also allows a fictitious header called `x-pbx-id`. This illustrates that the allowHeaders is a table of header names.

Script

```
M = {}
M.allowHeaders = {"History-Info", "x-pbx-id"}
function M.inbound_INVITE(msg)
    msg:convertHIToDiversion()
end
return M
```

Message

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

Result

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

getHeader

```
getHeader(header-name) returns header-value or nil
```

Given the string name of a header, this method returns the value of the header. For multiple headers with the same name, the values a concatenated together and comma separated:

Example: Set a local variable to the value of the Allow header.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local allow = msg:getHeader("Allow")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
.
```

```
Allow: UPDATE
Allow: Ack,Cancel,Bye,Invite
```

Output/Results

```
Local variable allow set to the string "UPDATE,Ack,Cancel,Bye,Invite"
```

getHeaderValues

```
getHeaderValues(header-name) returns a table of header values
```

Given the string name of a header, this function returns the comma separated values of the header as an indexed table of values. Lua indexing is 1 based: If there are n header values, the first value is at index 1 and the last value is at index n. Lua's **ipairs** function can be used to iterate through the table

Example: Create a local table containing the values of the History-Info header.

Script

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
    local history_info = msg:getHeaderValues("History-Info")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
.
History-Info: <sip:UserB@hostB?Reason=sip;cause=408>;index=1
History-Info: <sip:UserC@hostC?Reason=sip;cause=302>;index=1.1
History-Info: <sip:UserD@hostD>;index=1.1.1
```

Output/Results

```
Local variable history_info is  set to
{
  "<sip:UserB@hostB?Reason=sip;cause=408>;index=1",
  "<sip:UserC@hostC?Reason=sip;cause=302>;index=1.1",
  "<sip:UserD@hostD>;index=1.1.1"
}

In other words:
history_info[1] == "<sip:UserB@hostB?Reason=sip;cause=408>;index=1"
history_info[2] == "<sip:UserC@hostC?Reason=sip;cause=302>;index=1.1"
history_info[3] == "<sip:UserD@hostD>;index=1.1.1"
```

getHeaderValueParameter

```
getHeaderValueParameter(header-name, parameter-name) returns the parameter-value
```

Given the names of a header and a header parameter , this method returns the value of the specified header parameter. For multiple headers with same name, only the first header value is evaluated.

- If the header parameter is a tag=value format, the value on the right of the equal sign is returned.

- If the header parameter is a tag with no value, the value will be a blank string (i.e. "").
- If the header parameter does not exist in the first header value, nil will be returned.

Example: Set a local variable to the value of the header parameter named tag.

Script

```
M = {}
function M.outbound_180_INVITE(msg)
    local totag = msg:getHeaderValueParameter("To", "tag")
end
return M
```

Message

```
SIP/2.0 180 Ringing
.
To: <sip:1234@10.10.10.1>;tag=32355SIPpTag0114
```

Output/Results

```
Local variable totag is set to the string "32355SIPpTag0114"
```

getHeaderUriParameter

```
getHeaderUriParameter(header-name, parameter-name) returns the URI parameter-value
```

Given the string name of a header, this method returns the value of the specified URI parameter. For multiple headers with same name, only the first header value is evaluated.

- If the URI parameter is a tag=value format, the value on the right of the equal sign is returned.
- If the URI parameter is a tag with no value, the value will be a blank string (i.e. "").
- If the URI parameter does not exist in the first header value, nil will be returned.

Example: Set a local variable to the value of the uri parameter named user.

Script

```
M = {}
function M.outbound_180_INVITE(msg)
   local userparam = msg:getHeaderUriParameter("To", "user")
end
return M
```

Message

```
SIP/2.0 180 Ringing
.
To: <sip:1234@10.10.10.1;user=phone>;tag=32355SIPpTag0114
```

Output/Results

```
Local variable userparam is set to the string "phone"
```

addHeader

```
addHeader(header-name, header-value)
```

Given the string name of a header and a value, this method appends the value at the **end** of the list of header values.

Example: Add INFO to the Allow header.

Script

```
M = {}
function M.outbound_INVITE(msg)
    msg:addHeader("Allow", "INFO")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
Allow: Ack,Cancel,Bye,Invite
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
Allow: Ack,Cancel,Bye,Invite,INFO
```

addHeaderValueParameter

```
addHeaderValueParameter(header-name, parameter-name [,parameter-value])
```

Given a header-name, parameter-name, and optional parameter value, this method locates the specified header in the SIP message and adds the specified header parameter and optional parameter value to the header. The original header is replaced by the modified header value in the SIP message. If there are multiple header values for the specified header, only the first header value is modified. No action is taken if no instance of the specified header is located in the SIP message.

Example: Add the color=blue header parameter to the outbound Contact header.

Script

```
M = {}
function M.outbound_INVITE(msg)
    msg:addHeaderValueParameter("Contact", "color", "blue")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
Contact: <sip:1234@10.10.10.2>
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
Contact: <sip:1234@10.10.10.12>;color=blue
```

addHeaderUriParameter

```
addHeaderUriParameter(header-name, parameter-name [,parameter-value])
```

Given a header-name, parameter-name, and an optional parameter-value, this method locates the specified header in the SIP message and adds the specified header URI parameter and optional URI parameter value to the contained URI. The original header is replaced by the modified header value in the SIP message. If there are multiple header values for the specified header, only the first header value is modified. No action is taken if no instance of the specified header is located in the SIP message or if a valid URI is not found within the first instance of the specified header.

Example: Add user=phone parameter to the P-Asserted-Identity uri.

Script

```
M = {}
function M.inbound_INVITE(msg)
    msg:addHeaderUriParameter("P-Asserted-Identity", "user", "phone")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1;user=phone>
```

modifyHeader

```
modifyHeader(header-name, header-value)
```

Given the string name of a header and a value, this method over writes the existing header value with the value specified. This is effectively like invoking removeHeader followed by addHeader.

Example: Remove the display name from the P-Asserted-Identity header in outbound INVITE messages.

Script

```
M = {}
function M.outbound_INVITE(msg)
    -- Remove the display name from the PAI header
    local pai = msg:getHeader("P-Asserted-Identity")
    local uri = string.match(pai, "(<.+>)")
    msg:modifyHeader("P-Asserted-Identity", uri)
```

```
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" <1234@10.10.10.1>
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <1234@10.10.10.1>
```

applyNumberMask

```
applyNumberMask(header-name, mask)
```

Given a header-name and number mask this method locates the specified header in the SIP message and applies the specified number mask to the URI contained within the header. If a URI is successfully parsed from the header value, the number mask is applied to the user part of the parsed URI. The original header is replaced by the modified header value in the SIP message. If there are multiple instances of the specified header, only the first instance of the header is modified.

No action is taken if any of the following conditions exist.

1. No instance of the specified header is located in the SIP message
2. A valid URI is not found within the first instance of the specified header
3. The specified mask parameter is an empty string

Application of the number mask

The mask parameter defines the transformation to be applied to the user part of the header URI. Wildcard characters are specified in the mask parameter as an upper case X. For example, if the mask "+1888XXXXXXX" is specified, "X" is used as the insertion character of the mask. If this mask is applied to the example user part "4441234", the resulting sting is "+18884441234".

If the number of characters found in the user part to be masked is less than the number of wildcard characters in the mask, the left most wildcard characters will left as "X". Applying the previous mask to the example user part "1234" yields the resulting string "+1888XXX1234". If the number of characters found in the user part to be masked is greater than the number of wildcard characters in the mask, the left most characters of the user part are truncated. For example, if the mask "+1888XXXX" is applied to the user part "4441234", the resulting string is "+18881234".

Example: Apply a number mask the number in the P-Asserted-Identity header for inbound INVITE messages.

Script

```
M = {}
function M.inbound_INVITE(msg)
    msg:applyNumberMask("P-Asserted-Identity", "+1919476XXXX")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:+19194761234@10.10.10.1>
```

convertDiversionToHI

```
convertDiversionToHI()
```

Cisco Unified CM supports sending the Diversion header for handling redirecting information. For example, when a call is forwarded by Cisco Unified CM. In such instances, the SIP message may contain a Diversion header which is used by Cisco Unified CM to send the redirecting number. However, many SIP servers use the History-Info header for this purpose instead of the Diversion header. This API is used to convert the Diversion header into History-Info headers.

Note Some implementations require use of the raw header APIs (e.g. getHeader, addHeader, modifyHeader, and removeHeader) instead of or in addition to this API.

Example: Convert Diversion headers into History-Info headers for outbound INVITE messages. Then remove the Diversion header

Script

```
M = {}
function M.outbound_INVITE(msg)
    if msg:getHeader("Diversion")
    then
        msg:convertDiversionToHI()
        msg:removeHeader("Diversion")
    end
end
return M
```

Message

```
INVITE sip:2400@10.10.10.2 SIP/2.0
.
Diversion: <sip:1002@10.10.10.1>;reason=unconditional;privacy=off;screen=yes
```

Output/Results

```
INVITE sip:2400@10.10.10.2 SIP/2.0
.
History-Info: <sip:1002@10.10.10.1?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:2400@10.10.10.2>;index=1.1
```

convertHIToDiversion

```
convertHIToDiversion()
```

Cisco Unified CM supports receiving the Diversion header for handling redirecting information. For example, when a call is forwarded before reaching Cisco Unified CM. In such a case, the SIP message may contain a Diversion header which is used by Cisco Unified CM to establish the redirecting number. However, many SIP servers use the History-Info header for this purpose instead of the Diversion header.

Note Some implementations may require use of the raw header APIs (e.g. getHeader, addHeader, modifyHeader, and removeHeader) instead of or in addition to this API.

Steve L - Following note added for CSCub77698 - Sep2012

Note A Diversion header will only be added if the History-Info header has a "cause=" tag in it. In th e example below, note that there is only one Diversion header, but two History-Info headers, one of which has the “cause=” tag.

Example

The allowHeaders must specify "History-Info". Without it, the incoming History-Info headers will get dropped before script processing. Thus the call to convertHIToDiversion won't produce any useful results.

Script

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
    msg:convertHIToDiversion()
end
return M
```

Message

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

Result

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

removeHeader

```
removeHeader(header-name)
```

Given the string name of a header, this method removes the header from the message.

Example: Remove the Cisco-Guid header from outbound INVITE messages.

```
M = {}
function M.outbound_INVITE(msg)
    msg:removeHeader("Cisco-Guid")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" <1234@10.10.10.1>
Cisco-Guid: 1234-4567-1234
Session-Expires:  1800
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" <1234@10.10.10.1>
Session-Expires:  1800
.
```

removeHeaderValue

```
removeHeaderValue(header-name, header-value)
```

Given the string name of a header and a header-value, this method removes the header-value from the message. If the value was the only header-value, the header itself is removed.

Example: Remove X-cisco-srtp-fallback from the Supported header for outbound INVITE messages.

Script

```
M = {}
function M.outbound_INVITE(msg)
    msg:removeHeaderValue("Supported", "X-cisco-srtp-fallback")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" 1234@10.10.10.1
Supported: timer, replaces, X-cisco-srtp-fallback
Session-Expires:  1800
```

Output/Results

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" 1234@10.10.10.1
Supported: timer, replaces
Session-Expires:  1800
```

Manipulating SDP

getSdp

```
getSdp() returns string
```

This method returns the SDP as a Lua string. It returns nil if the message does not contain SDP.

Example: Establish a local variable which contains the SDP (or nil if there is no SDP).

Script

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
end
return M
```

Message

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.
Max-Forwards: 69
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

Output/Results

The variable sdp is set to the following string. Note that this string will contain embedded carriage return and line feed characters (i.e. "\r\n").

```
v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
```

```
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

setSdp

```
setSdp(sdp)
```

This method stores the specified string in the SIP Message as the SDP content body. The Content-Length header of SIP message is automatically updated to the length of the string.

Example: Remove the a=ptime lines from the outbound SDP.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        -- remove all ptime lines
        sdp = sdp:gsub("a=ptime:%d*\r\n", "")
        -- store the updated sdp in the message object
        msg:setSdp(sdp)
    end
end
return M
```

Message

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.
Max-Forwards: 69
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

Output/Results

The Final SIP message after the execution of above script will look like below:

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.
Max-Forwards: 69
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

removeUnreliableSdp

```
removeUnreliableSdp()
```

This method removes SDP from a 18X message. Before removing the SCDP, it checks to ensure the messages does not require PRACK.

Note The adjustments to the Content-Length header are made automatically when this API results in the SDP actually being removed. The Content-Type header is automatically removed if there is no other content body in the message.

Example: Remove SDP from inbound 180 messages.

Script

```
M = {}
function M.inbound_180_INVITE(msg)
    msg:removeUnreliableSdp()
end
return M
```

Message

```
SIP/2.0 180 Ringing
.
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
```

```
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

Output/Results

The Final SIP message after the execution of above script will look like below:

```
SIP/2.0 180 Ringing
.
Content-Length: 0
```

Manipulating Content Bodies

- getContentBody
- addContentBody
- removeContentBody

getContentBody

```
getContentBody(content-type) returns the content-body, content-disposition,
content-encoding, and content-language of the specified content-type
```

Given the string name of a content-type, this function returns the content-body, content-disposition, content-encoding, and content-language. The content-disposition, content-encoding, and content-language are optional and may not have been specified in the message. If the particular header is not specified in the message, nil is returned for its value.

If the content-body is not in the message, nil values are returned for all returned items.

Example: Establish local variables containing the content information for an outbound INFO message.

Script

```
M = {}
function M.inbound_INFO(msg)
    local b = msg:getContentBody("application/vnd.nortelnetworks.digits")
end
return M
```

Outbound Message

```
INFO sip: 1000@10.10.10.1 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.57:5060
From: <sip:1234@10.10.10.57>;tag=d3f423d
To: <sip:1000@10.10.10.1>;tag=8942
Call-ID: 312352@10.10.10.57
Cseq: 5 INFO
Content-Type: application/vnd.nortelnetworks.digits
Content-Length: 72

p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
```

```
d=4
```

Output/Results

```
Local variable b is set to the string:
p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
d=4
```

addContentBody

```
addContentBody(content-type, content-body [,content-disposition [,content-encoding
[,content-language]]])
```

Given the content-type and content-body, this API will add the content body to the message and make the necessary changes to the content-length header. The script may optionally provide disposition, encoding, and language as well.

Example: Add a plain text content body to outbound UPDATE messages.

Script

```
M = {}
function M.outbound_UPDATE(msg)
    msg:addContentBody("text/plain", "d=5")
end
return M
```

Message Before Normalization

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 0
```

Message After Normailization

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 3
Content-Type: text/plain
d=5
```

removeContentBody

```
removeContentBody(content-type)
```

Given the content-type, this API will remove the associated content body from the message and make the necessary changes to the content-length header.

Example: Remove the text/plain content body from outbound UPDATE messages.

Script

```
M = {}
```

```
function M.outbound_UPDATE(msg)
    msg:removeContentBody("text/plain")
end
return M
```

Message Before Normailization

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 3
Content-Type: text/plain

d=5
```

Message After Normailization

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 0
```

Blocking Messages

- block

block

```
block()
```

This method can block an inbound or outbound unreliable 18X message. For inbound blocking, CUCM behaves as if it never received the message. For outbound message blocking, CUCM doesn't send the message to the network.

Note If PRACK is enabled and therefore the message is actually a reliable 18X, CUCM will effectively ignore the call to block(). The message will not be blocked and an error SDI trace will be produced.

Example: Block outbound 183 messages.

Script

```
M = {}
function M.outbound_183_INVITE(msg)
    msg:block()
end
return M
```

Message

```
SIP/2.0 183 Progress
Via: SIP/2.0/TCP 172.18.199.186:5060;branch=z9hG4bK5607aa91b
From: <sip:2220@172.18.199.186>;tag=7cae99f9-2f13-4a4d-b688-199664cc38a4-32571707
To: <sip:2221@172.18.199.100>;tag=3affe34f-6434-4295-9a4b-c1fe2b0e03b6-21456114
.
.
```

Output/Results

```
The 183 message is not sent to the network.
```

Transparency

- getPassThrough

getPassThrough

```
getPassThrough() returns a per message pass through object
```

This method returns a SIP Pass Through object. It can be invoked for inbound messages. Information added to the pass through object is automatically merged into the outbound message generated by the other call leg. The automatic merging of this information takes place before outbound normalization (if applicable).

Refer to the SIP Pass Through APIs for use of the pass through object and examples.

Maintaining Per-Dialog Context

- getContext

getContext

```
getContext() returns a per call context
```

This method returns a per call context. The context is a Lua table. The script writer can store and retrieve data from the context across various message handlers used throughout the life of the dialog. The script writer does not need to be concerned with releasing the context at the end of the dialog. Cisco Unified CM will automatically release the context when the dialog ends.

Example:

This script uses context to store the History-Info headers received in an INVITE. When the response to that INVITE is sent, the stored headers are retrieved and added to the outgoing responses. The "clear" parameter is used to prevent these headers from being appended to subsequent reINVITE responses.

Script

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
    local history = msg:getHeader("History-Info")
    if history
    then
        msg:convertHIToDiversion()
        local context = msg:getContext()
        if context
        then
            context["History-Info"] = history
        end
```

```
    end
end

local function includeHistoryInfo(msg, clear)
    local context = msg:getContext()
    if context
    then
       local history = context["History-Info"]
       if history
       then
           msg:addHeader("History-Info", history)

           if clear
           then
             context["History-Info"] = nil
           end
       end
    end
end

function M.outbound_18X_INVITE(msg)
    includeHistoryInfo(msg)
end

function M.outbound_200_INVITE(msg)
    includeHistoryInfo(msg, "clear")
end
return M
```

Message Before Normalization

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1

SIP/2.0 180 Ringing
.
Content-Length: 0

SIP/2.0 200 OK
.
Content-Type: application/sdp
```

Message After Normalization

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1


SIP/2.0 180 Ringing
.
Content-Length: 0
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1,
<sip:1001@10.10.10.1>;index=1.1


SIP/2.0 200 OK
.
Content-Type: application/sdp
```

```
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1,
<sip:1001@10.10.10.1>;index=1.1
.
```

Utilities

isInitialInviteRequest

```
isInitialInviteRequest() returns true or false
```

This method returns true if the message is a request, the method is INVITE, and there is no tag parameter in the To header. Otherwise, false is returned.

Example:

Set a local variable based on whether or not the outbound INVITE is an initial INVITE. In this example, the INVITE is an initial INVITE and therefore, the value would be true for this particular INVITE.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local isInitialInvite = msg:isInitialInviteRequest()
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>
```

Output/Results

```
Local variable isInitialInvite set to 'true'
```

Example:

Set a local variable based on whether or not the outbound INVITE is an initial INVITE. In this example, the INVITE is not an initial INVITE and therefore, the value would be false for this particular INVITE.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local isInitialInvite = msg:isInitialInviteRequest()
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

```
.
To: <sip:1234@10.10.10.1>;tag=1234
```

Output/Results

```
Local variable isInitialInvite set to 'false'.
```

isReInviteRequest

```
isReInviteRequest() returns true or false
```

This method returns true if the message is a request, the method is INVITE, and there is a tag parameter in the To header. Otherwise, false is returned.

Example:

Set a local variable based on whether or not the outbound INVITE is a reINVITE. In this example, the INVITE is a reINVITE and therefore, the value would be true for this particular INVITE.

Script

```
M = {}
function M.outbound_INVITE(msg)
    local isReInvite = msg:isReInviteRequest()
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;tag=112233445566
.
```

Output/Results

```
Local variable isReInvite set to 'true'.
```

Example:

Set a local variable based on whether or not the outbound INVITE is a reINVITE. In this example, the INVITE is not a reINVITE and therefore, the value would be false for this particular INVITE.

Script

```
M = {}
function M.outbound_ANY(msg)
    local isReInvite = msg:isReInviteRequest()
end
return M
```

Message

```
CANCEL sip:1234@10.10.10.1 SIP/2.0
  .
```

Output/Results

```
Local variable isReInvite set to 'false'.
```

getUri

```
getUri(header-name) returns a string or nil
```

Given a header-name, this method locates the specified header in the SIP message and then attempts to retrieve a URI from that header. If there are multiple instances of the specified header, the URI of the first instance of the header is returned. Lua nil is returned if no instance of the specified header is located in the SIP message or if a valid URI is not found within the first instance of the specified header.

Example: Set a local variable to the uri in the P-Asserted-Identity header.

Script

```
M = {}
function M.inbound_INVITE(msg)
    local uri = msg:getUri("P-Asserted-Identity")
end
return M
```

Message

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
.
```

Output/Results

```
Local variable uri is set to "sip:1234@10.10.10.1"
```