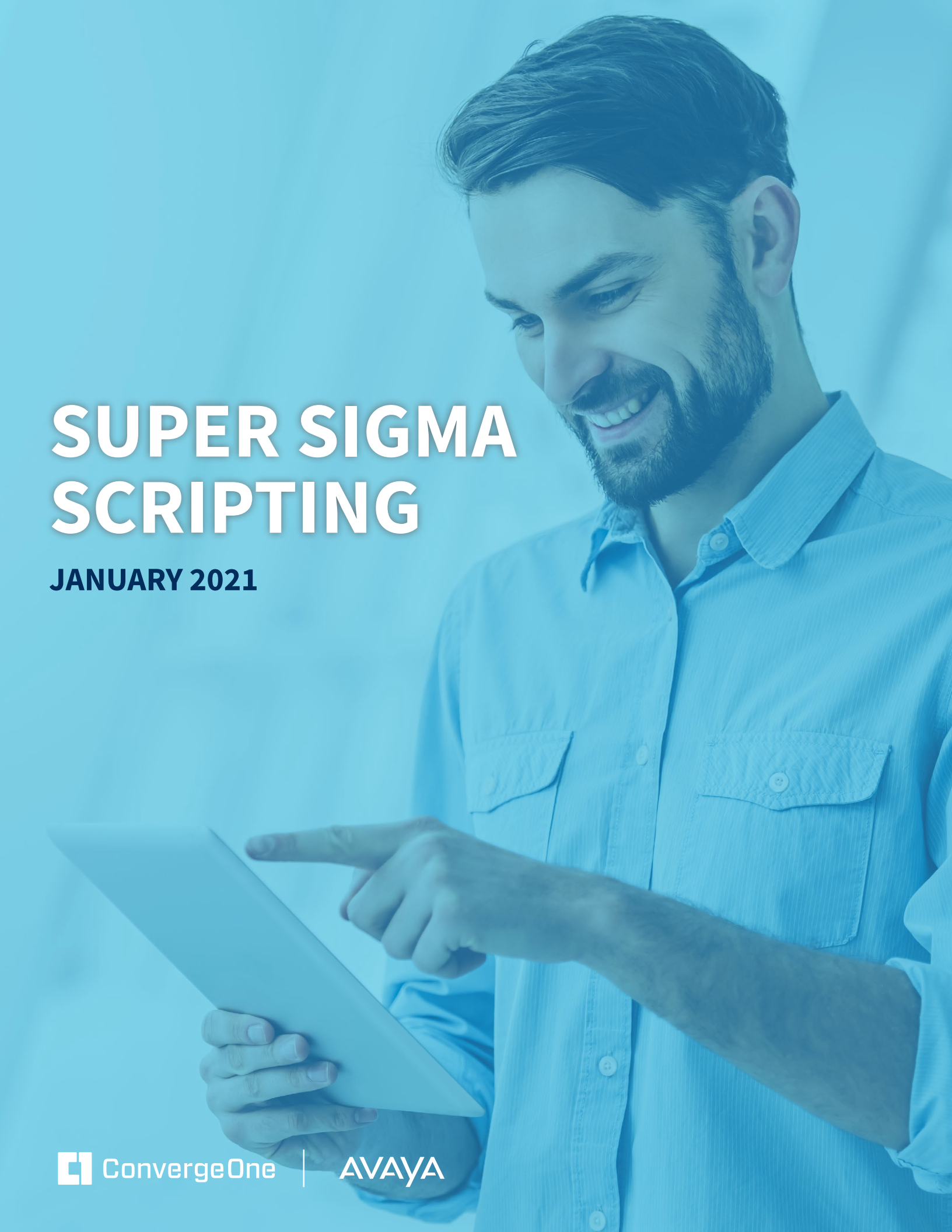


SUPER SIGMA SCRIPTING

JANUARY 2021

ConvergeOne | AVAYA

Having the ability to create a script that performs sophisticated manipulation of SIP and SDP messages is a super power of Avaya's Session Border Controller Enterprise (SBCE). Signaling Manipulation (SigMa) scripting gets its power by borrowing concepts from Regular Expressions (RegEx), a sophisticated pattern-matching tool invented in the 1950s.

This guide begins with an overview of SigMa and finishes with examples that illustrate using Avaya's proprietary SigMa scripts to solve common compatibility issues. SigMa is granular enough to work its magic on just Requests, just Responses, or both types of messages. It is designed to manipulate Session Description Protocol (SDP) as well.

The script can be created externally as a regular text file using a utility like WordPad and imported in the Signaling Manipulation screen. Alternatively, the script can be written directly in the SBCE using the embedded Sigma Editor.

TABLE OF CONTENTS

Section One: Why You Need a SigMa Script

For simple compatibility, virtually every SIP Proxy is expected to be able to manipulate some aspects of the SIP message. One reason to adapt messages is to standardize the information so that routing rules are simpler. For example, converting all telephone numbers to the E.164 scheme. The other common reason to adapt messages is so that information will be understood by the recipient.

I identify four types of adaptations. The first is to modify the user-portion of the SIP URI (Uniform Resource Identifier) address. Recall that a SIP address is the form of sip:user-portion@domain, for example sip:+19524563604@convergeone.com. Typically, one modifies the telephone number appearing in the user-portion into or out of E.164 format. This allows making routing decision based on a consistent quantity of digits. Later, after the routing destination has been determined, an adaptation might remove digits so the URI matches what the recipient expects, such as a voice mail system designed for 5-digit mailboxes.

Similarly, the domain-portion of the address can be converted between an IP address (sip:1234@192.168.1.55) and a SIP Domain "name" (sip:1234@convergeone.com). Generally, Avaya servers prefer a domain-name in the URI, while nearly every other vendor wants a domain-IP. Further, some Avaya servers, such as Communication Manager, pay close attention to the domain, thereby encouraging the use of sub-domain-names to sort incoming SIP requests into the appropriate CM signaling-group.

Translating a message into the appropriate SIP "Dialect" constitutes another adaptation. The issue is that "SIP" is not a single RFC (Request for Comment) standard. Since its ratification in 2001, "SIP" as we now understand it has been improved by the addition of about a hundred fully-ratified RFCs (and a few that were not ratified).

With so many choices, it’s common that the set of SIP standards implemented by one vendor does not match the set chosen by another vendor. So, the SIP messages have to be “translated” so that in a session each device receives messages it understands how to process.

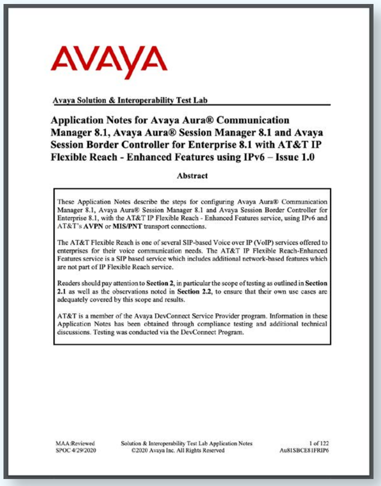
Frequently, information needs to be removed from SIP messages. There are three reasons for this type of adaptation. First, some headers need to be removed because they are too revealing. That could be because they provide the software version down to a patch level (e.g. User-Agent or Server), or it could be because they reveal the actual identity of the caller when the customer wants the caller-id to provide a generic public identity (“Helpdesk@convergeone.com). Another reason could be to remove information that might be confusing because it only has value within the network that created it and leads to misinterpretation in a different system.

The biggest reason to remove “unneeded” headers is to ensure they don’t jeopardize delivery of the critical headers. A few devices, such as Selta SIP Endpoints, cannot accept a packet if its total size, including overhead and payload, exceeds 2,000 bytes.

But the more common reason to remove useless SIP headers concerns the Transport protocol. Within Ethernet networks, the Maximum Transmission Unit (MTU) defines the maximum amount of data payload that can be contained within a single frame/packet. The MTU is not much of an issue with Transmission Control Protocol (TCP) because if the data doesn’t fit within one packet, it can be sent within additional packets. In contrast, with User Datagram Protocol (UDP) if the data doesn’t fit within one packet, the excess is simply discarded. This is a common problem with SIP Service Providers who have traditionally used UDP for their SIP trunks to the Public Switched Telephone Network (PSTN). With the MTU set to 1,500 Bytes, and a SIP request often starting at 1,200 Bytes, there is only room for about 300 Bytes of growth. So removing unneeded headers is common practice.

Recent versions of Avaya’s Aura Session Manager (ASM) have almost equaled the SBCE’s ability to manipulate SIP messages. I read once that Avaya advocates performing most adaptions at the edge of the network (where the SBCE lives), and leaving only the minimum of manipulations to be performed at the center of the network (where ASM lives). However, the SBCE is a highly critical security device and so administrator access to its menus should be extremely controlled. When contrasted with the freer access to ASM, that restricted access to the SBCE makes troubleshooting adaptations more difficult. So, I recommend the opposite of Avaya. Perform as many adaptations as possible within ASM, leaving the remainder for the SBCE.

Avaya documentation asserts: “Use of the Signaling Manipulation scripts require higher processing requirements on the Avaya SBCE. Therefore, this method of header manipulation should only be used in cases where the use of Server Configuration Profiles or Signaling Rules does not meet the desired result.”



AVAYA

Avaya Solution & Interoperability Test Lab

Application Notes for Avaya Aura® Communication Manager 8.1, Avaya Aura® Session Manager 8.1 and Avaya Session Border Controller for Enterprise 8.1 with AT&T IP Flexible Reach - Enhanced Features using IPv6 – Issue 1.0

Abstract

These Application Notes describe the steps for configuring Avaya Aura® Communication Manager 8.1, Avaya Aura® Session Manager 8.1 and Avaya Session Border Controller for Enterprise 8.1, with the AT&T IP Flexible Reach - Enhanced Features service, using IPv6 and AT&T’s **AVPN** or **MIS/PNT** transport connections.

The AT&T Flexible Reach is one of several SIP-based Voice over IP (VoIP) services offered to enterprises for their voice communication needs. The AT&T IP Flexible Reach-Enhanced Features service is a SIP based service which includes additional network-based features which are not part of IP Flexible Reach service.

Readers should pay attention to **Section 2**, in particular the scope of testing as outlined in **Section 2.1** as well as the observations noted in **Section 2.2**, to ensure that their own use cases are adequately covered by this scope and results.

AT&T is a member of the Avaya DevConnect Service Provider program. Information in these Application Notes has been obtained through compliance testing and additional technical discussions. Testing was conducted via the DevConnect Program.

MAA:Reviewed SPOC 4/29/2020 | Solution & Interoperability Test Lab Application Notes ©2020 Avaya Inc. All Rights Reserved | 1 of 122 Au81SBCE81FRIP6

AVAYA SOLUTION + INTEROPERABILITY TEST LAB

These Application Notes describe the steps for configuring Avaya Aura® Communication Manager 8.1, Avaya Aura® Session Manager 8.1 and Avaya Session Border Controller for Enterprise 8.1, with the AT&T IP Flexible Reach - Enhanced Features service, using IPv6 and AT&T’s AVPN or MIS/PNT transport connections.

Click Here to Access the Document

Section Two: Apply a SigMa script to a Server or a Flow?

If looking for a granular solution, the administrator can apply a SigMa script to just a single Server Flow or to several flows. Recall that the SBCE allows up to 32 Server Flows. Or for wider effect, the script could be applied to all traffic to/from a particular server through Server Configuration Profile.

The rule is that a script can only be applied to either a Flow or a Server, but not both. If you mistakenly assign a script to both a Flow and to a Server, then only the script in the Flow will be processed because it has higher priority. The script assigned to the Server would be ignored.

You are permitted to assign a script to a Server Flow and to a Server Configuration Profile but only if they point to different destinations (e.g. A PSTN Proxy Server Flow and an ASM1 Server Configuration Profile).

Each year, Avaya writes a couple dozen Application Notes detailing how to integrate the SBCE with a SIP Service Provider's (e.g. AT&T, Bell Canada, TELUS, and Verizon) SIP trunk offer. Having reviewed nearly all Application Notes from the last couple years, I can summarize them as follows:

- Every one of them applied the script to the Server Configuration Profile, not the Server Flow. Specifically, they assigned the SigMa script to the "Trunk Server", namely the carrier's proxy server.
- All of the scripts had the SBCE invoke adaptations POST_ROUTING. Just a few of those scripts also invoked adaptations at either PRE_ROUTING or AFTER_NETWORK.

Section Three: Simplified Anatomy of a Script

Reducing SigMa to its basics, it consists of just a few things. It has Filters that focus on certain messages, Actions that invoke a change, and Variables that contain data. To distinguish sub-processes from major processes Sigma uses nesting Brackets. Because scripts might be hard to interpret, the script can contain Comments. This section is an overview, followed by detailed descriptions of the options.

1. Filters

SigMa provides four levels of filtering.

- The **"Within Session"** filter identifies whether to limit attention to only some messages (e.g. just INVITEs) or to pay attention to all messages within a Session.
- The **"Act On… where"** filters identify:
  - which Requests/Responses/Messages to consider
  - which direction (Inbound / Outbound)
  - which hook point (AFTER_NETWORK, PRE_NETWORK, or POST_NETWORK)
  - Can add additional filters by using AND / OR

2. Actions

This is where the contents of a message can be added to, removed or changed. A statement can assign some value to a Variable. Or a statement can be a "conditional" in the format of "IF (some value = true) THEN {perform a function} ELSE {perform a different function}". And/or it could use a RegEx construct to match a value when performing an action. Finally to aide in troubleshooting, the SBCE can be told to write information to the process log file, although I have never found an example of PRINT being used on a production system.

3. Variables

Variables are easily identified because they always start with a percent sign (%). SigMa allows you to create your own variables by prepending the percent symbol to any alpha-numeric name. The only “special character” a variable name can include is an underscore (%variable_1). However, SigMa provides built-in variables and arrays for the most commonly referenced elements within a message (%HEADERS). Note that the built-in headers are fully UPPERCASE characters, so it is common practice to avoid all-uppercase when creating your own variables.

4. Curly-Brackets for Nesting

On the line immediately below the “within session ….” statement, you must begin with open-curly-bracket ({). The brackets are used to nest a specific set of filters and/or function statements as subordinate to some other statements above it.

To assist interpretation, the brackets and statements are indented with each nesting, and out-dented when that set of statements is completed. The structure follows the pattern of a traditional Microsoft Word outline, where major points are left-justified, sub-points are indented, and sub-sub-points are further indented.

Other points:

- Each curly-bracket should be on a line by itself. No other text or functions should appear on that line. This is a recommendation, not a rule.
- Each-and-every open-bracket must be paired with a closed-bracket (}). This is a rule.

5. Quotes for Values

When matching for alpha-numeric values, the string needs to be enclosed within double-quote marks (“) (for example “P-Asserted-Identity”). Matching is not case-sensitive, meaning the value would match (“P-ASSERTED-IDENTITY” as well as “p-asserted-identity”)

6. Semi-Colon for Actions

Each function statement that performs an action must end with a semi-colon (;) so that subsequent statements are not misinterpreted.

Note in this example that the curly-brackets (and the indentation) dictate that if the matching condition is met (the URI phone number is 1-952-456-3604) then two actions (assigning a new URI phone number and a Display Name) will be performed

Example

```
if(%HEADERS["From"][1].URI.USER = "19524563604")then

{

%HEADERS["From"][1].URI.USER = “18005551212”;

%HEADERS["From"][1].DISPLAY_NAME = “C1 Help Desk”;

}
```

7. Forward-Slashes for Comments

So that another engineer can understand what your script is intended to do, you would be wise to include comments describing your code. A line that begins with a double forward-slash (//) or slash-star (/*) indicates a comment, and that line will be ignored by the SigMa interpreter.

Section Four: Filter - "Within Session"

A "session" equates to a SIP Dialog (for example, starting with initial INVITE and ending immediately after the 200 OK response to a BYE request). Usually, the "within session" is a wide-open filter, looking at "ALL" Request and Response messages. Often it is limited to just those messages associated with an INVITE. Rarely is it limited to the messages associated with a REGISTER or other Requests.

Examples

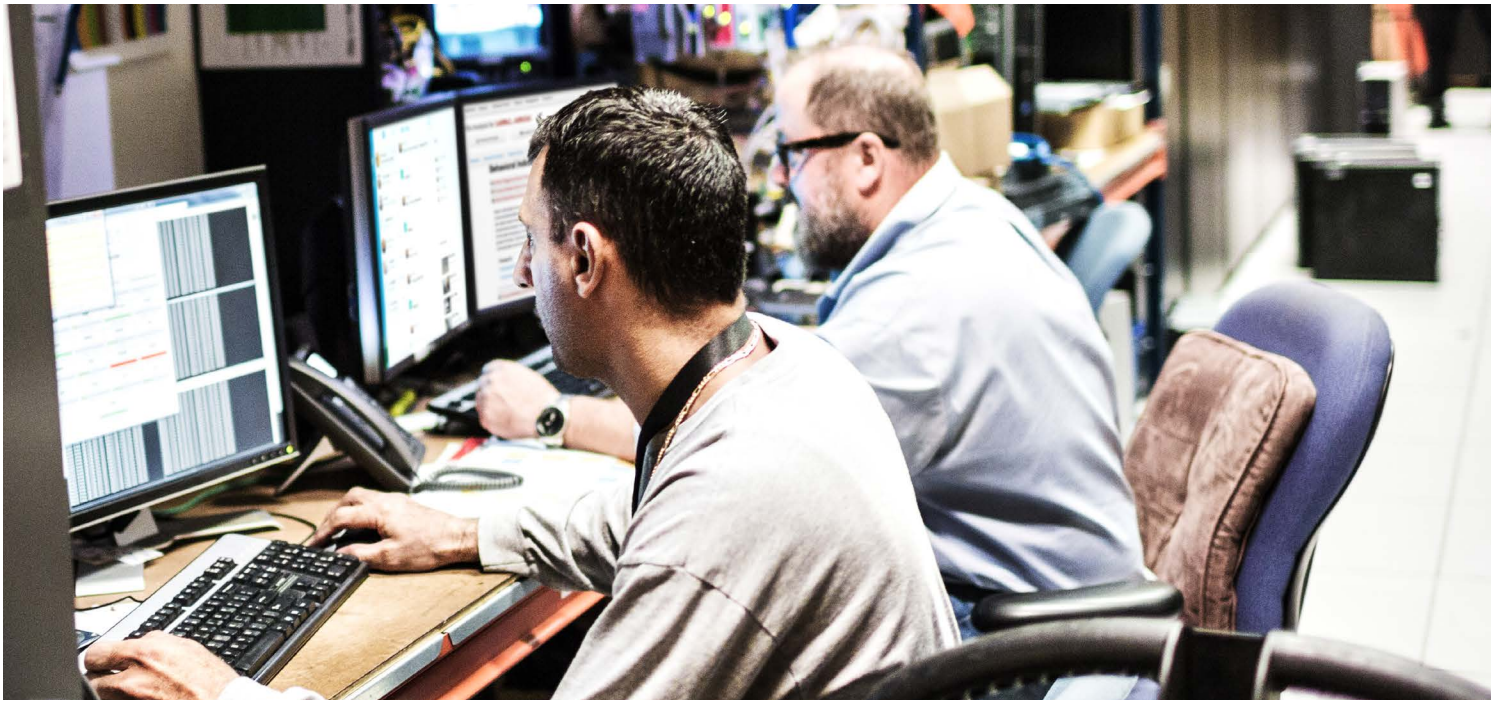
```
within session "ALL"

within session "INVITE"

within session "REGISTER"
```

Section Five: Filter - "Act On"

Usually the "act on" filter is administered to inspect all messages. Often it is limited to just Requests, and only occasionally is it set to "act on" just Responses.

Examples

```
act on message

act on request

act on response
```



Section Six: Filter - “Where Direction= and Entry_Point=”

This filter follows the “act on” filter to further narrow the scope. The two choices for direction are INBOUND and OUTBOUND, which are similar to Session Manager’s concept of Ingress and Egress. In other words, INBOUND means an incoming Request to the SBCE and that Request could be coming in from either the untrusted (PSTN) or trusted (ASM) interfaces.

Closely associated to direction are the choice of “Entry_Point” (sometimes called Hook_Point). There are three choices of “Hook Point”:

AFTER_NETWORK

This point occurs immediately after the message has been received from the network and approved by the firewall, but before any attempt to match the message to a Flow. A script applied at this point can modify all the incoming messages coming from a particular server and so is often used to normalize the message to simplify matching to a Flow. Consequently, using this hook point in a script means the script can only be assigned in a Server Configuration Profile configuration.

PRE_ROUTING

Like the AFTER_NETWORK point, this point occurs just before the SBCE attempts to match a message to a flow. However, instead of manipulating a message coming from a particular server, this modifies messages from all sources.

POST_ROUTING

This point occurs after the message has been matched to a flow, but before it has left the SBCE. So, often the modifications done here are to make the message acceptable to the intended recipient (e.g. ASM).

In the table below, each row lists the only acceptable combinations of Direction and Entry_Point.

Direction	Entry_Point
INBOUND	AFTER_NETWORK
INBOUND	PRE_ROUTING
OUTBOUND	POST_ROUTING

Examples

```
where %DIRECTION="INBOUND"  and %ENTRY_POINT="AFTER_NETWORK
where %DIRECTION="INBOUND"  and %ENTRY_POINT="PRE_ROUTING
where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING
```

Section Seven: Filter - Extra Modifiers

By adding modifiers such as AND/OR with other variables, the filter can narrow the focus. The other Session Variables that can be used on this line include the following;

Variable	Sample Values	Works only with	Description
%METHOD	INVITE PRACK SUBSCRIBE MADEUP		Matches on any Request, even those that are proprietary.
%REQ_METHOD	INVITE PRACK SUBSCRIBE	Act on response	Like %METHOD, but only looks at the responses.
%RESP_CODE	180 Ringing 301 Moved	Act on response	Matches on a particular numeric SIP response code in the 100 to 699 range.
%IN_DIALOG	TRUE or FALSE		Returns “TRUE” if the message is part of an existing dialog, or “FALSE” if it is starting a new dialog *(A reader reported to me that this variable does not appear to work.)*

Examples

```
where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK" and %METHOD="INVITE"

where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING" and %METHOD="ACK"

where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK" and %RESP_CODE="200"

where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK" and %RESP_CODE="180"
or %RESP_CODE="183" or %RESP_CODE="200"
```

Section Eight: In-Message Variables

The previous section identified variables used when looking across many SIP messages. In this section we identify the variables used when inspecting a single message.

The two most important families of built-in variables are the %HEADERS for working with SIP headers, and %SDP for working with Session Description Protocol (SDP) parameters.

Which SIP Header

To match the name of a particular SIP header, enclose it in double-quote marks surrounded by square-brackets ([ ]). The match is not case-sensitive. For example to match the CONTACT header, type the following:

```
%HEADERS[“Contact”][1]
```

However with SDP, the whole SDP section counts as a single message body. While in theory it might be possible to have multiple SDP messages bodies piggybacking on a SIP message, in reality never more than one is present. So the format will always begin as:

```
%SDP[1]
```

For SDP, capturing the whole message body has very little practical value. Instead, look below for how to pick a particular parameter.

Header Instance [ ]

In a SIP message some headers will be repeated, each time with a different value. For example, in a message you might encounter a dozen or more VIA, ROUTE, and/or RECORD-ROUTE headers. To pick a particular instance of the header, the syntax is: %HEADERS[“<Header-name>”][<Header position>]. For most headers, such as FROM and Contact, the Header Position is always 1. For a few headers, like VIA , and ROUTE, the positions can range from 1 to a large integer.

```
remove(%HEADERS["History-Info"][3]);

remove(%HEADERS["History-Info"][2]);

remove(%HEADERS["History-Info"][1]);
```

Header Parameters

Most SIP headers list several parameters, so SigMa has a system for retrieving all of them or just a particular parameter. To retrieve all the parameters in a P-Asserted-Identity use the following:

```
%HEADERS[“P-Asserted-Identity”][1]
```

To retrieve only the domain from the caller’s URI address in the P-Asserted-Identity, use the following:

```
%HEADERS[“P-Asserted-Identity”][1].URI.HOST
```

Similarly, with SDP there is a way to retrieve parameters. For example, you might want to identify the third codec (also known as: “format” or “payload type”). To do so, you start by specifying the 1st “m=” field, which identifies the media (i.e. audio, video, text, application, message, image, or control) port, and offered codecs. Then the script returns the value of the 3rd codec/format.

```
%SDP[1]["s"]["m"][1].FORMATS[3]
```

Other In-Message Variables

SigMa provides some other variables that draw information from within a SIP message.

Message Body

The message body at the bottom of a SIP NOTIFY request might contain XML (eXtensible Markup Language) text that needs modification. For example to replace a particular extension (4563604) with an anonymous identity, use the following:

```
%BODY[1].regex_replace("sip:4563604@convergeone.com","sip:anonymous@
anonymous.invalid");
```

Initial Request

You might want to test if this is the initial INVITE within a session (as opposed to a re-INVITE) and then take some (unspecified) action

```
if (%INITIAL_REQUEST = "true") then
```

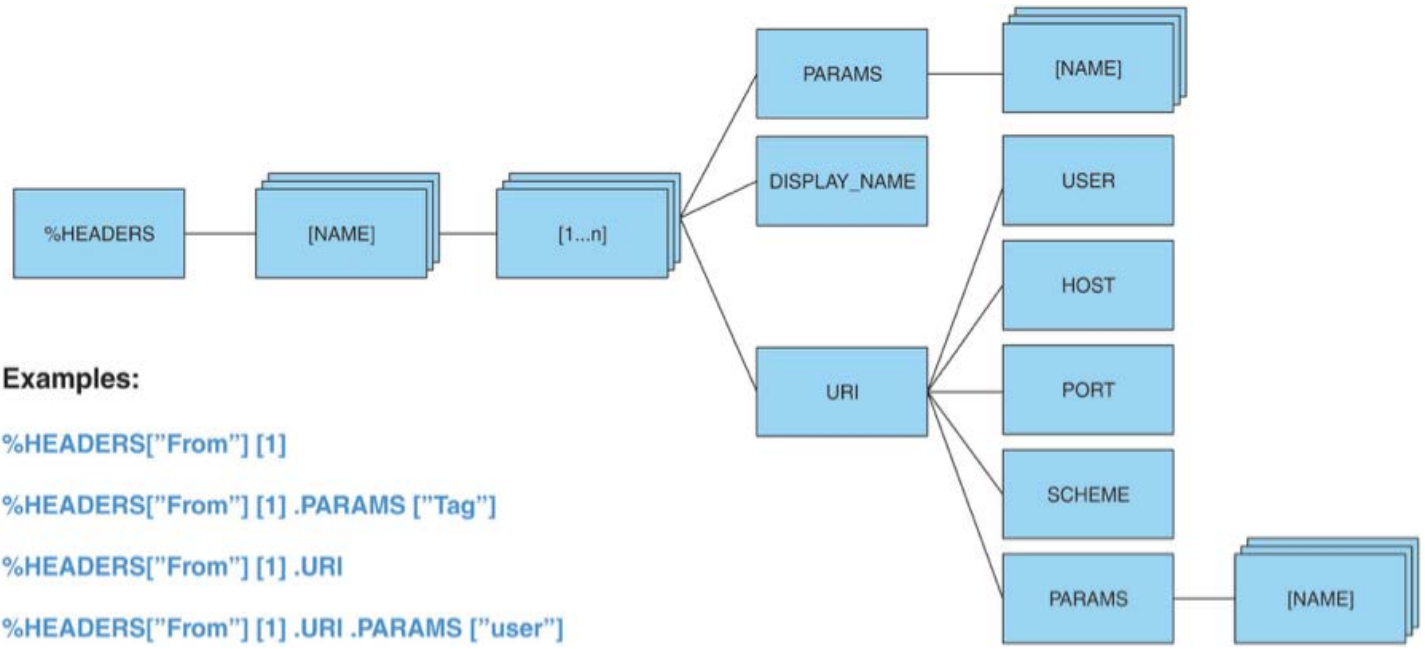
Remote IP

Might want to use the remote IP address to construct a Diversion header.

append(%HEADERS["Diversion"][1], %REMOTE_IP);

Variable	Valid Forms	Description
%HEADERS[]	%HEADERS[“Name”][n]	Used to retrieve an entire header. The second dimension ‘n’ denotes the nth instance of the header in the message. Value of n can be 1...∞
	%HEADERS[“Name”][n].PARAMS[“Name”]	Used to retrieve parameters within a header.
	%HEADERS[“Name”][n].DISPLAY_NAME	Refers to the display name within a header.
	%HEADERS[“Name”][n].URI	Refers to the URI within a header.
	%HEADERS[“Name”][n].URI.USER, %HEADERS[“Name”][n].URI.HOST, %HEADERS[“Name”][n].URI.PORT, %HEADERS[“Name”][n].URI.SCHEME, %HEADERS[“Name”][n].URI.PARAMS[“Name”]	Refers to various elements within a URI.

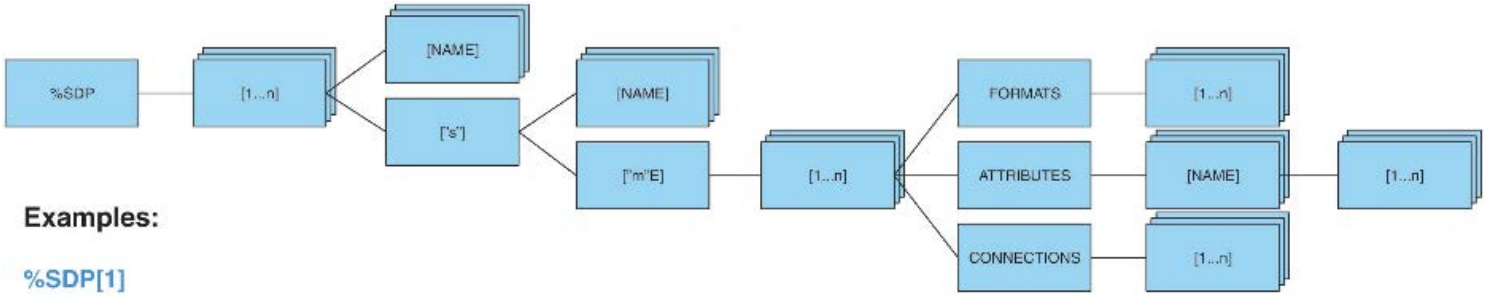
The following graphic, borrowed from Avaya’s SBCE Administration Guide, might assist you in specifying an element within a SIP Header.



The following table explains how to specify elements within SDP parameters.

Variable	Valid Forms	Description
%SDP[]	%SDP[n]	Refers to an entire nth SDP specification. Index n can be 1…∞.
	%SDP[n][“Name”]	Refers to a header within an SDP.
	%SDP[n][“Name”] [“SessionHdrName”]	Refers to a session header (like media) within an SDP session.
	%SDP[m][“s”][“m”][n]	Refers to nth media specification.
	%SDP[l][“s”][“m”][n].FORMATS[n]	Refers to nth media format specification.
	%SDP[j][“s”][“m”][k]. ATTRIBUTES[“Name”][n]	Refers to nth instance of “Name” attribute in the kth media specification.
	%SDP[m][“s”][“m”][n]. CONNECTIONS[k]n]	Refers to the kth connection from nth media specification.

The following graphic, borrowed from Avaya’s SBCE Administration Guide, might assist in specifying elements within SDP parameters.



Examples:

%SDP[1]

%SDP[1] ["v"]

%SDP[1] ["s"] ["t"]

%SDP[1] ["s"] ["m"] [k] . FORMATS[1]

%SDP[1] ["s"] ["m"] [k] . ATTRIBUTES{"fmtp"] [1]

cysbvasv LAO 021413

Section Nine: Action - Assign a Value to a Variable

It's common to want to transfer the information found in one header to another header. For example some SIP endpoints allow the user to input any Display Name (e.g. Mickey Mouse), which the endpoint uses to populate the FROM header. When the phone's INVITE routes through Aura Session Manager, ASM uses the administered name (e.g. John Waber) to populate the P-Asserted-Identity header. An adaptation might be needed to use the validated identity in the P-Asserted-Identity to overwrite the bogus identity in the FROM header.

The syntax is to begin with the variable name, use the equal sign (=) to assign a value, and end with a semi-colon (;). For example:

```
%var = "1";
```

The value might be statically assigned as in the preceding example, or it might be picked up dynamically from another variable which pulled the value from the SIP message. The example below transfers the value from the P-Asserted-Identity header to the FROM header in two steps.

```
%from_user = %HEADERS["P-Asserted-Identity"][1];
%HEADERS["From"][1] = %from_user;
```

This example directly transfers the value from the TO header to the Request_Line.

```
%HEADERS["Request_Line"][1].URI.USER = %HEADERS["To"][1].URI.USER;
```

Section Ten: Matching Condition

Often you will want to modify a message but only if some condition is true. That's where the following conditional function comes in handy.

If(condition is met) then{do function} else{do a different function}

Other protocols use a single equals-sign (=) to assign a value and a double equals-sign (==) to check for a match. However, SigMa uses a single equals-sign (=) both to assign a value and to check for a match.

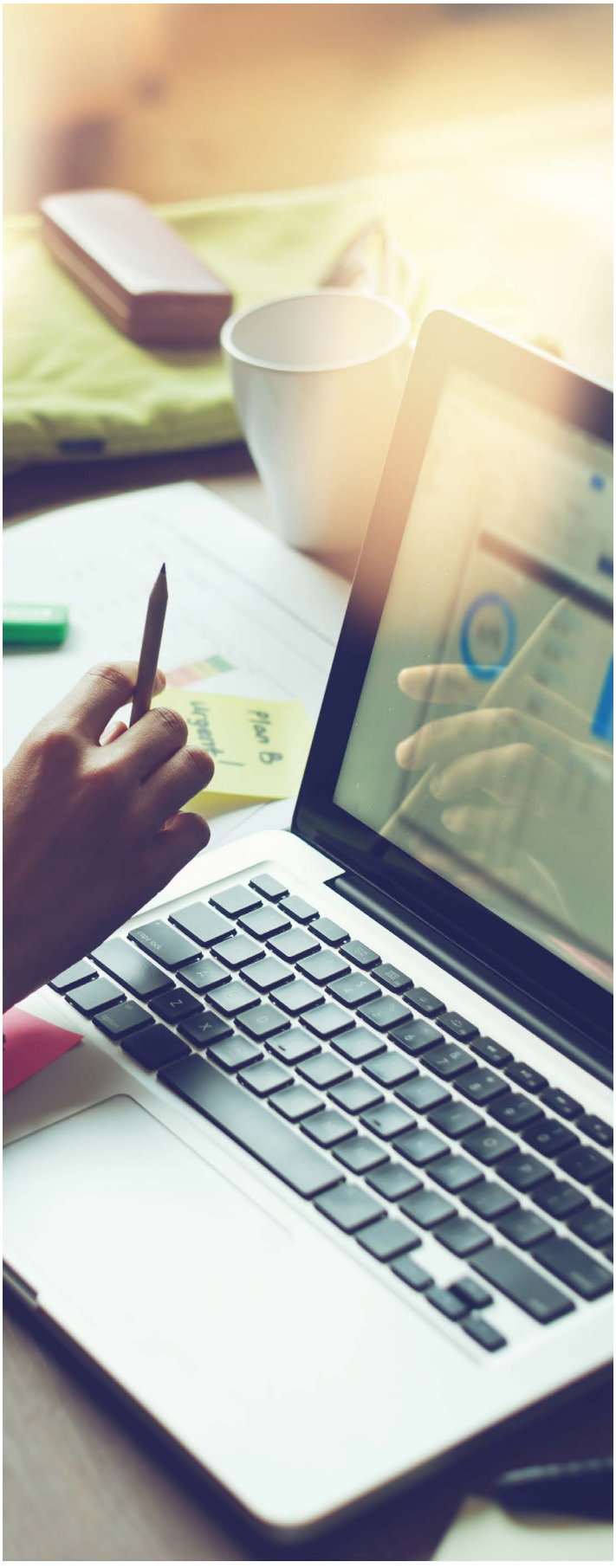
```
if(%HEADERS["Privacy"][1] = "none")
then

{

%HEADERS["Privacy"][1] = "id";

}
```

To look for a non-match, prepend an exclamation-mark to the equals-sign (!=), or to the function.

```
if(!exists(%HEADERS["Content-Type"]
[1])) then

{

%HEADERS["Content-Type"]
[1]="application/sdp";

}
```

Notes:

- The "else" is not required and is often omitted.
- The matching condition is enclosed in parentheses (), and the action function is enclosed with curly-brackets {}.
- You may nest an if()then {}else{} function within another if()then {}else{} function



To increase the capability of the “Matching Condition,” SigMa offers two matching functions: “Exists()” and regex_match(). Both functions return a “TRUE” if the item listed between the parentheses is found in the message (or a “FALSE” if it’s not found). The difference is that “Exists()” is limited to looking only for the presence of a particular header or a header’s parameter.

```
if (exists(%HEADERS["Contact"][1].PARAMS["+avaya-cm-keep-mpro"]))then
{
remove(%HEADERS["Contact"][1].PARAMS["+avaya-cm-keep-mpro"]);
}
```

In contrast, Regex_match() uses the power of Regular Expressions for a more nuanced search pattern. In the following example, the goal is to remove the OPTIONS parameter from the Allow header. The challenge is that a missing coma or an extra comma might jeopardize processing the SIP message. So, Regex_match() checks if “OPTIONS” is last on the list of parameters, somewhere in the list, or the only parameter in the list.

```
if(%HEADERS["Allow"][1].regex_match(", OPTIONS"))then
{
%HEADERS["Allow"][1].regex_replace(", OPTIONS","");
}
else
{
if(%HEADERS["Allow"][1].regex_match(" OPTIONS,"))then
{
%HEADERS["Allow"][1].regex_replace(" OPTIONS,", "");
}
else
{
if(%HEADERS["Allow"][1].regex_match(" OPTIONS"))then
{
remove(%HEADERS["Allow"][1]);
}
}
}
```

Explaining Regular Expressions is beyond the scope of this document. However, here is a brief description of the "Special Characters" used in constructing Regular Expressions.

RegEx Meta-characters	Description	What it does
.	Period or Dot	Matches exactly one alpha-numeric character
*	Star or asterisk	Matches any quantity of alpha-numeric characters, including no matches at all.
+	Plus	Modifies a search to ensure there is at least one, but could be more, matches
!	Exclamation	Negation ("Not"), or do not match this
?	Question mark	Makes the preceding character optional. (So 'sips?' matches 'sip' and 'sips')
|	Vertical bar or pipe	Indicates "or", and can be used several times when matching multiple elements
{n}	Curly brackets	Match the preceding item exactly n times
[ ]	Square brackets	Match on any of characters between the brackets
^	Caret	Matches starting at the beginning of a line
$	Dollar Sign	Although standard RegEx uses the dollar sign ($), that symbol **cannot** be used in a SigMa Regular Expression. (It would have meant start looking for a match at the end of the line)

If you need to match on the literal meaning of any of these special characters, you need to disable its power with a backslash (/). For example if you want to match **1+1=2**, the correct regex is **1\+1=2**. Otherwise, the plus sign has a special meaning.

However, the backslash in front of a literal characters (i.e. not a special character) can create a RegEx token with a special meaning. Some of the common examples of RegEx tokens are listed below

Token	Meaning
\d	Matches one digit 0-9, making it a shorthand for [0-9]
\n	Matches one “new-line” character (i.e. Line-Feed, or <LF>)
\r	Matches one “carriage-return,” character (i.e. <CR>)
\s	Matches one “white-space” character
\t	Matches one “tab” character
\w	Matches one “word” (consisting of numbers, characters, or the underscore)

Below are some of the Regular Expressions I have found in SigMa scripts.

Expression	Meaning ("IF")
"transport=tcp;”	If text exactly matches what’s contained between the quotation marks in “transport=tcp;”
“, OPTIONS”	If OPTIONS is preceded by a coma and a space
“sip: 2911@192.168.3.150"	If URI exactly matches that address
“^10”	If number begins (indicated by ^) with 10 (and can have any quantity of digits/characters after ‘10’)
"1800(.*)”	If URI begins with 1800. The dot (.) and star (*) are wildcards with the dot matching a single digit (0-9) and the star matching any quantity of digits, including no digits (null). Their combination (dot-star) is frequently used to ensure there is at least something that follows, but whatever it is can be of any quantity of alpha-numeric characters.
"a=inactive\r\n"	If text matches “a=inactive”. Usually, RegEx stops at the first match. But the (\r\n) compels RegEx to not stop at the first match but to keep looking on other lines for more matches.
"^.....”	If the parameter begins (indicated by ^) exactly five digits/characters (as indicated by five dots)

"b=(TIAS|AS|CT):(\d+)\r\n"	Find all bandwidth statements in SDP. The vertical-bars (“pipe) mean “OR” so it will search for TIAS or AS or CT. The backslash-d-plus (/d+) means find one-or-more (+) digits (\d). The backslash-r-backslash-n (\r\n) means to keep looking for additional matches on other lines.
"\+"	If the parameter contains a plus-sign (+). Because plus-signs have significance in RegEx, the backslash in front of it disables the special meaning. So in this case, RegEx looks for a literal plus-sign.
"^13xxxxxx"	If the number begins (indicated by ^) with 13 followed by six digits (indicated by the six ‘x’ digit-wildcards)
"613xxx650[6-8]”	If the 10-digit number begins with 613 and ends with 6506, 6507 or 6508. The square-brackets define the set of choices, namely 6 through 8. If the digits were separated by a comma [6,8], it would limit the set of choices to just 6 or 8.
"origlocname=\"(.+?)\"	Match the contents of the origlocname= parameter, capturing everything within the quotes (\” \”). The (.+?) expression re-quires matching on at least one character. (In contrast, the (.*?) expression can also match on no characters)
"^[0-9]{5}?"	The entry starts with a five-digit number. The carrot says to the entry has to start with the value. The square-brackets constrain the digits to be from 0 to 9. The curly-brackets indicate that five instances of the constrained value can exist.

Section Eleven: Retrieving Information: regex_get()

Occasionally, you may want to retrieve specific information from a message so you can store it in a variable. You can use Regular Expressions within the parenthesis to search for the information. If the expression matches content, then the function will return a string. If no match is found, the string will be empty. To use Regular Expression to find text anywhere within a header, use the function: HEADERS[“Header”].regex_get(“regex”)

Example

```
%var_loc = %HEADERS["P-Location"][1].regex_get("origlocname=\"(.+?)\"");
```

Similarly, to find text within a particular parameter of the header, use the function: %HEADERS[“Header”]. PARAMS[“Param”].regex_get(“regex”)

Section Twelve: Changing Information: regex_replace

To use Regular Expression to find text within a header and then replace it with different text, use the function: HEADERS[“Header”].regex_replace(“regex”, “new-text”)

Example

```
%HEADERS["Diversion"][1].regex_replace("sips","sip");
```

```
%BODY[1].regex_replace("a=inactive\r\n","a=recvonly\r\n");
```

Similarly, to find text within a header’s parameter, and then replace it, use the function: %HEADERS[“Header”].PARAMS[“Param”].regex_replace(“regex”, “string”)

Example

```
%HEADERS["To"][1].URI.USER.regex_replace("^.....","");
```

```
%HEADERS["From"][1].URI.USER.regex_replace("\+","");
```

Section Thirteen: Adding Information: append()

To append a string to a header use the function: append()

Example

```
append(%HEADERS["Diversion"][1], %REMOTE_IP);
```

Section Fourteen: Deleting Information: remove()

To remove a header and all of its parameters, use the function: remove(%HEADERS[“Header”])

Examples

```
remove(%HEADERS["P-Location"][1]);
```

```
remove(%BODY[1]);
```

To remove just a parameter from a header, use the function: remove(%HEADERS[“Header”].PARAMS[“Param”])

Examples

```
remove(%HEADERS["Contact"][1].URI.PARAMS["gsid"]);
```

```
remove(%HEADERS["Contact"][1].URI.PARAMS["epv"]);
```

Annotated Sample Scripts

The following are scripts culled from Application Notes and from recommendations by fellow ConvergeOne System Engineers and Architects. I am particularly indebted to Nitas Patthanakittikul, as well as Michael Besco and Tim Bosi.

I confess to adding a few lines in order to make the scripts more complete. Also, I show them as separate scripts but most production scripts are a combination of two or more of the scripts shown below. Finally, I have listed them by frequency of use, with the most commonly used scripts nearer the top of the list.

1. SIP Header Optimization: Removing Headers on outbound messages to PSTN

With SIP trunks, often you need to remove unneeded content from the messages leaving the organization and going to the PSTN. This could be because the trunks use UDP transport and there is the possibility that the message will exceed the 1500 Byte MTU limit of an Ethernet packet. Or you need to remove it because the header is too revealing about your environment.

What to Remove?

Avaya's proprietary SIP headers are prime candidates for removal because they have no value in the PSTN. In particular, the P-Location header is the most commonly targeted because it can contain an abundance of content, but add little value to outsiders.

In addition, Avaya inserts two headers that are so important that Avaya also copies their contents into yet another header. The issue is that those headers would be discarded if the SIP Request traversed another vendor's Back-to-Back-User-Agent (B2BUA), such as an SBC. So, Avaya appends the contents of the Av-Global-Session-ID (gsid) and the Endpoint-View (epv) to the Contact header, trusting that every vendor's SBC will dutifully copy the incoming Contact header verbatim into the outgoing message.

The History-Info headers track how a call has been redirected and are expendable once the message has left the enterprise. Most of the sample scripts assume that only three History-Info headers exist in a message which I'm not sure is a safe assumption.

Note that you can remove all these headers in a Signaling Rule, which Avaya documentation asserts is a better than removing them with a SigMa script. However, nearly every Avaya Application Note that describes how to integrate the SBCE with a carrier's SIP Trunk, uses a SigMa script to remove headers.

```
within session "ALL"

{

act on request where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"

{

//Remove unwanted Avaya headers.

remove(%HEADERS["Accept-Language"][1]);

remove(%HEADERS["Alert-Info"][1]);

remove(%HEADERS["Av-Attendant"][1]);

remove(%HEADERS["Av-Correlation-Id"][1]);
```

```
remove(%HEADERS["Av-Global-Session-ID"][1]);
remove(%HEADERS["Av-Secure-Indication"][1]);
remove(%HEADERS["Endpoint-View"][1]);
remove(%HEADERS["P-Av-Message-Id"][1]);
remove(%HEADERS["P-Charging-Vector"][1]);
remove(%HEADERS["P-Conference"][1]);
remove(%HEADERS["P-Location"][1]);
remove(%HEADERS["P-Rental"][1]);
remove(%HEADERS["Reason"][1]);
remove(%HEADERS["Remote-Address"][1]);
remove(%HEADERS["Remote-Party-Id"][1]);
remove(%HEADERS["x-nt-e164-clid"][1]);

//Remove headers that are too revealing about your environment
remove(%HEADERS["User-Agent"][1]);
remove(%HEADERS["Server"][1]);

//Remove gsid and epv parameters from the Contact header.
remove(%HEADERS["Contact"][1].URI.PARAMS["gsid"]);
remove(%HEADERS["Contact"][1].URI.PARAMS["epv"]);

// Remove unneeded call redirection headers
remove(%HEADERS["History-Info"][3]);
remove(%HEADERS["History-Info"][2]);
remove(%HEADERS["History-Info"][1]);

// OPTIONAL: A few carriers do not support the Session-Expires RFC
remove(%HEADERS["Min-SE"][1]);
remove(%HEADERS["Session-Expires"][1]);
}
}
```

2. Remove inbound headers going to ASM

Sometimes, SIP Service Provider include headers that might be confusing within your organization.

```
within session "All"
{
act on request where %DIRECTION="INBOUND" and %ENTRY_POINT="PRE_ROUTING"
{
// Remove unwanted inbound headers
remove(%HEADERS["Alert-Info"][1]);
remove(%HEADERS["Organization"][1]);
remove(%HEADERS["Resource-Priority"][1]);
}
}
```

3. Changing Max-Forwards from 0 to 70 from carriers

Service Providers need to know if the SBCE is in service, so they might send it an OPTIONS request as a heartbeat. But rather than respond to the OPTIONS request, the default behavior of an SBC is to forward the request into the organization. To ensure it is the SBC that responds, the carrier might send an OPTIONS request but set the Max-Forwards value to zero. That compels the SBC to respond with 483 Too Many Hops. Since the carrier received a predicted response to its request, it knows the SBC is alive.

Unfortunately, that behavior causes the SBCE's log file to overfill with "Max-Forwards" events, meaning that important log entries are lost. A partial solution is for the script to reset the Max-Forward value to 70. (The Max-Forwards value is usually set to 70, but the range of acceptable integers is quite large.) The other part of the solution is administer the SBCE to intercept OPTION requests and reject/respond to them with the expected 200 Okay.

To ensure the change is made early in the processing, before the SBCE has a chance to respond, the script needs to be invoked "After_Network".

Also, you should have an if-clause to ensure the change only applies if this is an initial-request. That way if the request were to fall into a routing loop, the Max-Forwards would eventually reach 0 and a proxy would cancel the request. Otherwise, with every loop through the SBCE, the Max-Forwards value would be reset to the previous value, meaning the request would wastefully consume network resources for a very long time.

Finally, the Max-Forwards header is mandatory and must be present within every request. The following sample script includes a step which tests if the header is found. That is a useless step, but I include it only because I found it in many scripts and wanted to call it out.

```
within session "OPTIONS"
{
act on request where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK"
{
if (%INITIAL_REQUEST = "true") then
{
if (exists(%HEADERS["Max-Forwards"][1])) then
{
%HEADERS["Max-Forwards"][1] = "70";
}
}
}
}
```

4. Generic Caller-ID

Typically, the FROM header provides the Caller-ID to the called-party. An administrator might want a range of phone numbers (1952-xxx-xxxx) to provide a generic caller-id. Perhaps they want the public number (888) 777-7280 shared rather than individual extensions numbers. The solution is to match the extension number and replace it.

A URI is represented as “<display_name>”<scheme>:<user>@<host>:<port>. (For example: “John Waber”sip:19524563604@convergeone.com:5060). To obscure the caller’s identity we need to extract the Display_Name, and the user portion of the address using the URI.USER parameter.

```
within session "INVITE"
{
act on request where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
//Check if the user portion of the From URI begins with 1-952)
if(%HEADERS["From"][1].URI.USER.regex_match("^1952xxxxxxx"))then
{
//Store the user’s URI and Display Name in a temporary variables
%OriginalFromUri = %HEADERS["From"][1].URI.USER;
%OriginalFromName = %HEADERS["From"][1].DISPLAY_NAME;
```

```
// Change the Display Name and URI to the generic identity.
%HEADERS["P-Asserted-Identity"][1].DISPLAY_NAME = "ConvergeOne";
%HEADERS["Contact"][1].DISPLAY_NAME = "ConvergeOne";
%HEADERS["From"][1].DISPLAY_NAME = "ConvergeOne";
%HEADERS["From"][1].URI.USER = "18887777280";
}
}
//We are not done. The script continues below…
```

When the responses comes back, we need to reset the URI USER and DISPLAY NAME back to the actual user's information. So, before the response is sent into the organization, it is checked if the URI.USER is 18887777280. If yes, then change it back to the original user's details.

```
within session "INVITE"
{
act on response where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK"
{
//Check if the user portion of the FROM header is 18887777280
if(%HEADERS["From"][1].URI.USER = "18887777280")then
{
// Use the temporary variables to Change the Display Name and URI
// back to the generic identity.
%HEADERS["From"][1].URI.USER = %OriginalFromUri;
%HEADERS["From"][1].DISPLAY_NAME = %OriginalFromName;
}
}
```

5. Subtract the Plus

The clear indication that a phone number is formatted for E.164 is the presence of a leading plus sign. All SIP carriers want the user-portion of the URI to be formatted as E.164, but some will reject the request if the URI contains a plus sign. This script removes the plus sign from all the places it is commonly found.

```
within session "ALL"
{
act on message where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
```

```
{
//Manipulate headers
%HEADERS["Request_Line"][1].URI.USER.regex_replace("\+","");
%HEADERS["To"][1].URI.USER.regex_replace("(\+)","");
%HEADERS["From"][1].URI.USER.regex_replace("\+","");
%HEADERS["P-Asserted-Identity"][1].URI.USER.regex_replace("(\+)","");
%HEADERS["Contact"][1].URI.USER.regex_replace("(\+)","");
%HEADERS["Diversion"][1].URI.USER.regex_replace("(\+)","");
%HEADERS["Refer-To"][1].URI.USER.regex_replace("(\+)","");
}
}
```

6. Stray Secure-SIP Schemes

When TLS/SRTP is used within the enterprise, the SIP headers include the SIPS URI scheme for Secure SIP. But many carriers do not support SIPS. So, on outbound calls to the PSTN the SBCE can automatically convert most of the header schemes from SIPS to SIP. However, the SBCE does not have a convenient way to convert the URI scheme in Diversion headers. Consequently calls that require a Diversion header, such as calls forwarded off-network, and EC500 calls ("Extension to Cellular"), would be rejected by the carrier unless this script is used.

```
within session "ALL"
{
act on request where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
//Changes the Diversion header scheme from SIPS to SIP. %HEADERS["Diversion"]
[1].regex_replace("sips","sip");
}
}
```

7. Remove unwanted XML

The Avaya Aura Experience Portal uses Extensible Markup Language (XML) when communicating between its elements. The XML information is sent in the Message Body at the bottom of a SIP message. (Some people confuse Message Body with SDP. While SDP is one of the most common things sent within the Message Body, XML and many other protocols can be contained in it.). The XML content has no value when it leaves the organization and should be removed to reduce the size of the SIP packet.

```
within session "ALL"
{
act on message where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
// Remove unwanted xml element information from the SDP in SIP messages
// sent to Service Provider.
remove(%BODY[1]);
}
}
```

8. Add User=Phone.

Some carriers need to be told to interpret the user-portion of the URI as a phone number, which is achieved by adding the parameter user=phone to the URI.

```
within session "ALL"
{
act on message where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
append(%HEADERS["Diversion"][1].URI,";user=phone");
append(%HEADERS["P-Asserted-Identity"][1].URI,";user=phone");
append(%HEADERS["From"][1].URI,";user=phone");
}
}
```

9. Modify P-Asserted-Identity

To reduce call spoofing, carriers inspect the P-Asserted-Identity header to confirm it contains a number the carrier associates with the customer, such as a Direct Inward Dial number. Otherwise, the carrier rejects the request with the response: 403 Forbidden.

If an inbound trunk call is transferred back to the carrier using a SIP INVITE then the original caller's phone number will appear in the FROM and P-Asserted Identity headers. This problem happens with some Extension to Cellular (EC500) call scenarios, a phone's Call-Forward Off-Net is activated, or when Avaya Aura Experience Portal (AAEP) performs a consultative call transfer to an off-network number. The SIP Server performing the call redirection (e.g. AAEP) will insert into the INVITE a Diversion header containing its own phone number.

The workaround is for the script to check for the presence of a Diversion header, and if so use the phone number in the Diversion header to replace the one in the P-Asserted-Identity header. Although it might not be needed, the script also copies the Display Name, and then deletes the Diversion header.

```
within session "INVITE"
{
act on request where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
if (exists(%HEADERS["Diversion"][1])) then
{
%phone_number = %HEADERS["Diversion"][1].URI.USER;
%display_name = %HEADERS["Diversion"][1].DISPLAY_NAME;
%HEADERS["P-Asserted-Identity"][1].URI.USER = %phone_number;
%HEADERS["P-Asserted-Identity"][1].DISPLAY_NAME = %display_name;
remove(%HEADERS["Diversion"][1]);
}
}
}
```

10. Remove Bandwidth Statements

Depending on the call flow, some Avaya SIP endpoints (e.g., 9641, 9621, and 9608 models) may generate various Bandwidth SDP statements such as b=TIAS:64000, b=CT:64, and b=AS:64. A few carriers have issues if they observe a bandwidth statement in the SDP. Since the bandwidth statements are optional, a simple fix is to remove them from SDP.

```
within session "ALL"
{
act on message where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
//Remove Bandwidth statements from SDP
%BODY[1].regex_replace("b=(TIAS|AS|CT):(\d+)\r\n”,””);
}
}
```

11. Replace Pilot Number with Destination Number

On inbound calls to the customer, some carriers put the trunk group's pilot number in the Request-URI and put the Destination number in the TO header. Avaya Aura expects the destination number, typically a DID number, in the Request-URI. So, the script copies the DID phone number (URI.USER) from the TO header into the Request-URI.

```
within session "ALL"
{
act on message where %DIRECTION="INBOUND" and %ENTRY_POINT="PRE_ROUTING"
{
// On inbound calls from PSTN, copy the phone number from the TO header
// to the Request-URI.
%HEADERS["Request_Line"][1].URI.USER = %HEADERS["To"][1].URI.USER;
}
}
```

12. Fixing Faxes

In this example, inbound INVITEs from the carrier to the enterprise included Fax T.38 attributes in the SDP, even for calls that were not Fax calls. Then, all of the customer's Avaya one-X Communicator softphones rejected the calls because the requests contained a media descriptor (i.e. T.38) the softphones do not support.

```
within session "ALL"
{
act on message where %DIRECTION="INBOUND" and %ENTRY_POINT="AFTER_NETWORK"
{
// This script fixes a codec-mismatch issue with Avaya one-X Communicator
// by removing T.38 fax attributes from ALL SIP messages coming from PSTN
remove(%SDP[1]["s"]["m"][2]); //
}
}
```

13. Change Min-SE Timer

For both the Session-Expires and Min-SE headers, the carrier required different timer values than those provided by Avaya Aura. The easiest way to fix this issue was to replace the minimum acceptable session-expires header on all outgoing calls.

```
within session "INVITE"
```

```
{
act on request where %DIRECTION="OUTBOUND" and %ENTRY_POINT="POST_ROUTING"
{
// Change the Min-SE Value from 1,800 to 360 seconds
if(exists(%HEADERS["Min-SE"][1])) then
{
%HEADERS["Min-SE"][1].regex_replace("1800", "360");
}
}
}
```

About ConvergeOne

Founded in 1993, ConvergeOne is a leading global IT service provider of collaboration and technology solutions. Thousands of enterprise and mid-market customers trust ConvergeOne with customer experience, cybersecurity, data center, enterprise networking, and unified communications solutions to achieve business outcomes. Our investments in cloud infrastructure and managed services provide transformational opportunities for customers to achieve financial and operational benefits with leading technologies.

About the Author

John Waber
Senior Technical Instructor, Center of Excellence
ConvergeOne
JWaber@convergeone.com

John Waber is a senior technical instructor at ConvergeOne, with over ten years teaching Avaya products, and is a regular presenter at the annual International Avaya Users Group (IAUG) conventions.