Oracle® Communications Session Border Controller

Header Manipulation Rules Guide

Release S-Cz10.1.0 - for Service Provider and Enterprise
G49664-01
April 2026

ORACLE

Oracle Communications Session Border Controller Header Manipulation Rules Guide, Release S-Cz10.1.0 - for Service Provider and Enterprise

G49664-01

Contents

About This Guide

The *HMR Resource Guide* describes the SIP manipulation language called Header Manipulation Rules (HMR).

This publication is used with Oracle Communications Session Border Controller and Oracle Enterprise Session Border Controller.

Documentation Set

The following table describes the documentation set for this release.

Document Name	Document Description
Acme Packet 3900 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 3900.
Acme Packet 4600 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 4600.
Acme Packet 4900 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 3950 and Acme Packet 4900.
Acme Packet 6350 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 6350.
Acme Packet 6400 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 6400.
Release Notes	Contains information about the current documentation set release, including new features and management changes.
Known Issues & Caveats	Contains known issues and caveats
Configuration Guide	Contains information about the administration and software configuration of the Service Provider Session Border Controller (SBC).
ACLI Reference Guide	Contains explanations of how to use the ACLI, as an alphabetical listings and descriptions of all ACLI commands and configuration parameters.
Maintenance and Troubleshooting Guide	Contains information about SBC logs, performance announcements, system management, inventory management, upgrades, working with configurations, and managing backups and archives.
MIB Guide	Contains information about Management Information Base (MIBs), Oracle Communication's enterprise MIBs, general trap information, including specific details about standard traps and enterprise traps, Simple Network Management Protocol (SNMP) GET query information (including standard and enterprise SNMP GET query names, object identifier names and numbers, and descriptions), examples of scalar and table objects.
Accounting Guide	Contains information about the SBC's accounting support, including details about RADIUS and Diameter accounting.

Document Name	Document Description
HDR Guide	Contains information about the SBC's Historical Data Recording (HDR) feature. This guide includes HDR configuration and system-wide statistical information.
Admin Security Guide	Contains information about the SBC's support for its Administrative Security license.
Security Guide	Contains information about security considerations and best practices from a network and application security perspective for the SBC family of products.
Platform Preparation and Installation Guide	Contains information about upgrading system images and any pre-boot system provisioning.
Call Traffic Monitoring Guide	Contains information about traffic monitoring and packet traces as collected on the system. This guide also includes WebGUI configuration used for the SIP Monitor and Trace application.
HMR Guide	Contains information about configuring and using Header Manipulation Rules to manage service traffic.
REST API	Contains information about the supported REST APIs and how to use the REST API interface.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

My Oracle Support

My Oracle Support (https://support.oracle.com) is your initial point of contact for all product support and training needs. A representative at Customer Access Support (CAS) can assist you with My Oracle Support registration.

Call the CAS main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. When calling, make the selections in the sequence shown below on the Support telephone menu:

1. Select 2 for New Service Request.
2. Select 3 for Hardware, Networking, and Solaris Operating System Support.
3. Select one of the following options:
   - For technical issues such as creating a new Service Request (SR), select 1.
   - For non-technical issues such as registration or assistance with My Oracle Support, select 2.

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

Emergency Response

In the event of a critical service situation, emergency response is offered by the Customer Access Support (CAS) main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/

index.html. The emergency response provides immediate coverage, automatic escalation, and other features to ensure that the critical situation is resolved as rapidly as possible.

A critical situation is defined as a problem with the installed equipment that severely affects service, traffic, or maintenance capabilities, and requires immediate corrective action. Critical situations affect service and/or system operation resulting in one or several of these situations:

- A total system failure that results in loss of all transaction processing capability
- Significant reduction in system capacity or traffic handling capability
- Loss of the system's ability to perform automatic system reconfiguration
- Inability to restart a processor or the system
- Corruption of system databases that requires service affecting corrective actions
- Loss of access for maintenance or recovery operations
- Loss of the system ability to provide any required critical or major trouble notification

Any other problem severely affecting service, capacity/traffic, billing, and maintenance capabilities may be defined as critical by prior discussion and agreement with Oracle.

Locate Product Documentation on the Oracle Help Center Site

Oracle Communications customer documentation is available on the web at the Oracle Help Center (OHC) site, http://docs.oracle.com. You do not have to register to access these documents. Viewing these files requires Adobe Acrobat Reader, which can be downloaded at http://www.adobe.com.

1. Access the Oracle Help Center site at http://docs.oracle.com.
2. Click **Industries**.
3. Under the Oracle Communications sub-header, click the **Oracle Communications documentation** link.
   The Communications Documentation page appears. Most products covered by these documentation sets appear under the headings "Network Session Delivery and Control Infrastructure" or "Platforms."
4. Click on your Product and then Release Number.
   A list of the entire documentation set for the selected product and release appears.
5. To download a file to your location, right-click the **PDF** link, select **Save target as** (or similar command based on your browser), and save to a local folder.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Revision History

This section provides a revision history for this document.

Date	Description
April 2026	• Initial release.

1

Header Manipulation Rules

Variances among SIP networks, like incompatible vendor deployments or disparate SIP services, can degrade SIP services or disrupt SIP operations. To resolve these variances, Oracle deploys Header Manipulation Rules (HMR), giving network administrators the ability to control SIP traffic by manipulating SIP messages.

HMRs permit the network administrator to:

- Insert, delete, or modify SIP headers or parameters
- Copy or move header or parameter values
- Rename parameter names
- Modify MIME bodies including SDP, XML and ISUP
- Change SIP-I/SIP-T ISUP messages, parameters, and fields
- Change message information when, for example, normalization is required
- Categorize and label specific message flows for special processing
- Capture information from a message and insert it into another message

The SBC can perform these actions based on the following:

- The type of SIP message (Request or Response)
- The type of Request (INVITE, REGISTER, etc.)
- The success or failure of a regular expression to match a header or parameter

HMR Fundamentals

HMR is a tool language based on rulesets, header rules, and element rules.

- Rulesets contain one or more header rules, as well as optional element rules that operate on specified header elements. They are applied to inbound or outbound traffic for a session agent, realm, or SIP interface.
- Header rules operate on specific headers. They can contain element rules, each of which specify the actions to perform for a given element of this header.
- Element rules perform operations on the elements of a header. Header elements include all subparts of a header, excluding the header name; for example, header value, header parameter, and URI parameter.

The SBC cannot dynamically perform validation as you enter rules. Use the ACLI **verify-config** command to confirm that the HMR configuration does not contain invalid or circular references.

- An invalid reference is a reference that points to a non-existing rule.
- A circular reference is a reference that creates an endless loop of manipulation actions.

Audience

This document is intended for those users who already understand the Oracle Communications Session Border Controller and the SIP protocol. In addition, Oracle recommends you become as HMR-savvy as possible by attending Oracle training courses prior to launching any HMR in production. You should be aware of all issues that might result from misinformed or misapplied HMRs.

When to Use HMR

HMR is a flexible, powerful tool. As such, Oracle recommend using it with utmost care. HMR should only be implemented in production networks once the HMRs and their applications have been rigorously tested in a lab environment. You want to ensure your HMRs work as you intend them before using them for your production network.

Oracle's Customer Support Team can assist you in developing HMRs for your network. Our customer support team can ensure that your HMR are constructed, configured, and applied properly, thereby guaranteeing your HMR achieves the result you want.

Managing HMR Impact on Performance

The following suggestions help manage HMR effect on performance.

- Use the pre-constructed manipulations and variable tags. They consume less processing and decrease the effect on performance.
- Include constructs and constrain the HMR to specific methods and messages. For example, you can limit effected methods or the length of a string match.
- Construct the HMR to only work on the traffic that matches your criteria, letting the remaining traffic pass untouched. (Unless you want to manipulate all traffic.)
- Take advantage of the test tools available on SBC to evaluate your HMRs.
- Administer the HMRs by using HMR export and import and reorder tools also available.
- Use logfiles to resolve issues.

Applying HMRs to Traffic

You can apply HMR rules to inbound or outbound traffic for session agents, realms, and SIP interfaces. The order of precedence is:

1. session agent
2. realm
3. SIP interface

A SIP manipulation applied to a session agent overrides the other two, and a SIP manipulation for a realm overrides one for a SIP interface.

> **Note**
>
> HMRs cannot be applied to associated agents.

> **Note**
>
> Oracle Communications Session Border Controller sends the Content-Length header in OPTIONS message to Session Agent that has HMR configured.

Outbound HMR

Outbound HMR rules are applied just before the SIP message is sent out by the SBC, after SIP-NAT processing. Any changes made by the HMR affects the message. Those changes are not overridden by the SBC, which means the SBC does not prevent the rules from breaking the protocol.

The rules are performed in a stateless manner. They do not store values across messages and they do not remember what they did in previous messages.

> **Note**
>
> You can work around the stateless behavior by having an inbound HMR copy the information needed to a private header, which then goes through the SBC. The outbound rule can then look for the header and act upon the information.

Inbound HMR

Inbound HMR rules are applied before most processing done by the SBC, but after some SIP parser processing is performed. The message's source is determined to decide which session agent, realm, or SIP interface it belongs to.

By default, the header rules are applied after the message is parsed; this verifies the message is well-formed and follows the specifications. This is necessary to securely perform any subsequent message processing, including HMR. An exception to this rule can be created by setting the `inmanip-before-validate` option. See "SIP Header Pre-Processing HMR" for more details.

Because inbound rules are applied before the message is completely processed by the SBC, you can use them to make the SBC perform specific actions outside of ordinary processing. For example, you can change the request-URI, add a Route header, or add a trunk group URI to make the SBC route the request on a different path.

Inbound rules are stateless. However, if the SBC is in B2BUA mode (its most common mode) it stores and remembers certain header values for later use in the dialog. If HMR changes them on inbound, the SBC later believes them to be the actual received values. There are a few exceptions to this with the following headers:

- To and From can be changed by HMR and are used when the message gets forwarded out another interface.
  But if they were for a new request message, the SBC remembers the original ones when it sends back 1xx-6xx responses. The previous hop that sent the new request inspects the responses and needs them to be identical based on SIP protocol rules. However, requests sent by the SBC back to the originator for the call, from the called to the caller, will not be automatically undone by the SBC as the responses were.
- Call-ID values are stored before HMR is applied and cannot be changed by HMR on inbound.

If a SIP INVITE is received for a new call, inbound HMR can change the To or From headers so that the next hop device gets the changed headers and the SBC stores them. But the 100 Trying, 180 Ringing, and 200 OK responses, for example, will use the original To and From values and not the HMR modified ones. If the called party later sends a Bye or re-Invite, back to the caller, the SBC will then use the HMR modified values it stored, which may or may not be correct.

Order of Header Rule Application

The SBC applies SIP header rules in the order you have entered them. This guards against the SBC removing data that might be used in the other header rules.

This ordering also provides you with ways to strategically use manipulations. For example, you might want to use two rules if you want to store the values of a regular expression. The first rule would store the value of a matched regular expression, and the second could delete the matched value.

In addition to taking note of the order in which header rules are configured, you must also configure a given header rule prior to referencing it. For example, you must create Rule1 with the action store for the Contact header before you can create Rule2 which uses the stored value from the Contact header.

HMR Store Actions and Boolean Results

Although HMR rulesets are stateless (they do not store values across messages nor remember what they did in previous messages), they can store strings for use within the same ruleset. Some header rules and element rules can store values that later header rules or element rules can use. Once the set of header rules and element rules in a SIP manipulation are performed, and the SIP manipulation is complete for the message, the stored values are forgotten.

Routing Decisions

Before routing the message, the SBC parses the ingress SIP message, ensuring the validity of the message's structure. After this parsing, the SBC applies the inbound header manipulation. You can use the inbound HMRs to modify the SBC's routing behavior if you want to increase the flexibility of the routing options.

An outbound HMR is the last processing the SBC performs on traffic before passing it back to the interface hardware. Knowing where this processing fits in helps you to know what state the traffic will be in before being processed by the outbound HMR. Outbound traffic is not subject to the screening functions performed by the hardware on inbound traffic.

Static and Dynamic HMR

You can manipulate the headers in SIP messages both statically and dynamically. You can edit response headers or the Request-URI in a request, and change the status code or reason phrase in SIP responses.

Static HMR

Static HMR lets you set up rules that remove and/or replace designated portions of specified SIP headers. The SBC can:

- Search headers for dynamic content or patterns with the header value. It can search, for example, for all User parts of a URI that begin with 617 and end with 5555 (e.g., 617...5555).
- Manipulate any part of a patterns match with any part of a SIP header. For example, 617 123 5555 can become 617 231 5555 or 508 123 0000, or any combination of those.

Dynamic HMR

SIP HMR lets you set up dynamic header manipulation rules that give the SBC complete control over alterations to the header value. Using regular expressions provides a high degree of flexibility for header manipulation. For example, you can search a specific URI when you do not know the value of the parameter, but want to use the matched parameter value as the header value. It also lets you preserve matched sections of a pattern, and change what you want to change.

Sample HMR

The following shows a complete HMR that manipulates To and From headers, changes the URI-host element, and hides IP topology. It is applied as outgoing for a realm. The HMR includes a built-in HMR variable $REMOTE_IP.

```
sip-manipulation
    name                                    NAT_IP
    description
    split-headers
    join-headers
    header-rule
        name                                    To
        header-name                             To
        action                                  manipulate
        comparison-type                         case-sensitive
        msg-type                                request
        methods
        match-value
        new-value
        element-rule
            name                                    To
            parameter-name
            type                                    uri-host
            action                                  none
            match-val-type                          ip
            comparison-type                         case-sensitive
            match-value
            new-value                               $REMOTE_IP
    header-rule
        name                                    From
        header-name                             From
        action                                  manipulate
        comparison-type                         case-sensitive
        msg-type                                request
        methods
        match-value
        new-value
        element-rule
            name                                    From
```

```
parameter-name
type                                    uri-host
action                                  none
match-val-type                          ip
comparison-type                         case-sensitive
match-value
new-value                               $LOCAL_IP
```

HMR Components

Each SIP manipulation ruleset contains one or more header rules and element rules for use as an inbound or outbound HMR ruleset. Generally, you set a header rule that will store what you want to match, and then you create subsequent rules that operate on this stored value.

Because header rules and element rules are applied sequentially, a given rule performs its operations on the results of all the rules previously entered. For example, if you want to delete a portion of a SIP header, you would create Rule 1 that stores the value for the purpose of matching, and then create Rule 2 that would delete the portion of the header you want removed. This prevents removing data that might be used in the other header rules.

Relationship Between Rulesets and Its Rules

The relationship between manipulation rules and manipulation rulesets is created once you load your configuration. The order in which you enter rulesets does not matter. It also means that the SBC cannot dynamically perform validation as you enter rules, so you should use the **verify-config** command to confirm your manipulation rules contain neither invalid nor circular references. Invalid references are those that point to SIP manipulation rules that do not exist, and circular references are those that create endless loops of manipulation rules being carried out over and over.

Ruleset Guidelines

Keep the following guidelines in mind when creating rulesets:

- One ruleset per inbound message
- One ruleset per outbound message
- Header or element rules can call another HMR
- An HMR can have multiple header rules
- A header rule can have multiple header rules

Ruleset Components

The following table lists ruleset components.

Component	Description
header-rule	Header rules form the basis of rulesets. Used to operate on one or more SIP headers within the SIP message; operations performed at this level work on the entire header value, excluding the label. Within a ruleset, each HR is performed in order. Typically one performs regular expression "store" action HRs before manipulation ones, although there are exceptions depending on the needs. There is no hard limit to the number of HR elements included in a ruleset, although in practical terms one would probably not configure thousands of them.
match-value	Used to perform a matching comparison to decide whether to store values, add a header, or delete a header. The type of matching comparison performed is based on the comparison-type field. If the match-value is left blank, the action is performed regardless. Therefore, if the header rule action is "delete", "add", or "manipulate", and the match value is left blank, the action will be performed on the header. If the header rule action is "store" and the match value is left blank, the SBC automatically stores everything, as if the match value were `.+` which means match at least one character, as many times as possible. Note that any whitespace after the first non-whitespace character is kept as well.
element-rule	Used to operate on specific portions of a SIP header, such as components of a URI value within the header or the parameters of the header; if the header value contains a URI, then this class operates only on the specified portion (i.e., URI user or header parameter); this class does not operate on headers with multiple header values.
mime-rule	Used to operate on any MIME part within a SIP message (SDP, test, or some other proprietary body type); used as a general facility to operate on the entire body as a single continuous string.
mime-header-rule	Used to operate on the SIP headers within a body part; the body part contains headers only when the MIME content is contained in a multi-part message; when used to operate on a MIME body that is not multi-part, then this class operates as through it were a header-rule.
mime-isup rule	Special type of mime-rule because it expects the MIME content of the specified body to be part of a valid binary ISDN User Part (ISUP) format.
isup-param-rule	Used to perform operations on the parameters contained in an ISUP body.
mime-sdp-rule	Special kind of mime-rule that is used to operate on the SDP MIME content of a SIP message; at this level, the rule operates on the entire SDP as a single contiguous string.
sdp-session-rule	Used to operate on only the session portion of the SDP content consists of all the characters starting from the beginning until the first media line.
sdp-media-rule	Used to operate on only a specific media portion of the SDP content; consists of all the characters starting from the beginning of the specified m-line until the next m-line or the end of the SDP.
sdp-line-rule	Used to operate on a single descriptor line within either the session or media portion of the SDP.

Guidelines for Header and Element Rules

Header rules and element rules share these guidelines:

- References to groupings that do not exist result in an empty string.
- References to element rule names alone result in a Boolean condition of whether the expression matched or not.
- A maximum of ten matches are allowed for a regular expression. Match 0 (grouping 0) is always the match of the entire matching string; subsequent numbers are the results for other groups that match.
- Rule names must start with a letter, and then can contain any number of letters, numbers, or underscores.
- All uppercase rule names are not allowed because this syntax is reserved for variables.
- To avoid being interpreted as a minus operator, dashes are not permitted in rule names.

Guidelines for Header Rules

Header rules guidelines include:

- Header names must be unique in a given HMR.
- Each header rule operates on one header.
- Multiple header rules can operate on the same header.
- Header rules can contain multiple element rules.

Guidelines for Element Rules

Element rule guidelines include:

- Element rule names must be unique within a header rule
- Each element rule operates on one component of the header
- Multiple element rules can operate on the same component

Duplicate Header Names

If more than one header exists for a configured header-name, the SBC stores each value in an array whose index starts at 0. To reference those values, use the syntax `$<header-name>[<index>]`.

Add a trailing `[<index>]` value after the header-name parameter to represent the specific instance of the header on which to operate. Additional stored header values are indexed in the order in which they appear within the SIP message, and there is no limit to the index. The SBC takes no action if the header does not exist.

Use index 0 to reference the first header. In addition to numerical values, possible index values are:

- * The SBC references all headers.
- ^ The SBC references the last stored header in the header rule.

Note that the header instance functionality has no impact on HMR's add action, and you cannot use this feature to insert headers into a specific location. Headers are added to the end of the list, except that Via headers are added to the top.

SIP Header Pre-Processing HMR

By default, the SBC performs in-bound SIP manipulations after it carries out header validation. Adding the **inmanip-before-validate** option in the global SIP configuration allows the SBC to perform HMR on received requests prior to header validation. Because there are occasional issues with other SIP implementations—causing invalid headers to be used in messages they send to the SBC—it can be beneficial to use HMR to remove or repair these faulty headers before the request bearing them is rejected.

When configured to do so, the SBC performs pre-validation header manipulation immediately after it executes the top via check. Inbound SIP manipulations are performed in order of increasing precedence: SIP interface, realm, and session agent.

The fact that the top via check happens right before the SBC carries out pre-validation header manipulations means that you cannot use this capability to repairs the first via header if it is indeed invalid. If pre-validation header manipulation were to take place at another time during processing, it would not be possible to use it for SIP session agents. The system learns of matching session agents after top via checking completes.

For logistical reasons, this capability does not extend to SIP responses. Inbound manipulation for responses cannot be performed any sooner that it does by default, a time already preceding any header validation.

To enable SIP header pre-processing:

1. Access the **sip-config** configuration element.

   ```
   ORACLE# configure terminal
   ORACLE(configure)# session-router
   ORACLE(session-router)# sip-config
   ORACLE(sip-config)#
   ```

2. **options**—Set the inmanip-before-validate parameter.

   ```
   ORACLE(sip-config)# options +inmanip-before-validate
   ```

   This value allows the SBC to perform pre-validation header manipulation in order of increasing precedence: SIP interface, realm, and session agent.

3. Save and activate the configuration.

Back Reference Syntax

You can use back reference syntax in the new-value parameter for header and element rules. Denoted by the use of `$1`, `$2`, `$3`, etc. (where the number refers to the regular expression's stored value), you can reference the header and header rule's stored value without having to use the header rule's name. It instead refers to the stored value of this rule.

For example, when these settings are in place:

- header-rule=changeHeader
- action=manipulate
- match-value=`(.+)([^;])`

you can set the new-value as `sip:$2` instead of `sip:$changeHeader.$2`.

You can use the back reference syntax when:

- The header-rule action parameter is set to manipulate or find-replace-all
- The element-rule action parameter is set to replace or find-replace-all

Using back reference syntax simplifies your development work because you do not need to create a store rule and then manipulate rule; the manipulate rule itself performs the store action if the comparison-type parameter is set to pattern-rule.

Dialog Matching

The `out-of-dialog` setting is useful for To/From NATing rules.

Service providers can use HMR to support legacy RFC 2543 devices and some non-compliant RFC 3261 devices. The header-rule msg-type setting called `out-of-dialog` has been added, which applies the rule (and any of its sub-rules) only to out-of-dialog requests. If the rule was applied as an outbound sip-manipulation to the first request, then it will apply the rule against all subsequent requests going in the same direction. The primary purpose of this new configuration setting is to support changing the To/From URI's in mid-dialog requests without breaking dialog matching for some over-strict SIP devices.

About Dialog-Matching Header Manipulations

The goal of this feature is to maintain proper dialog-matching through manipulation of dialog-specific information using HMR. Two fundamental challenges arise when looking at the issue of correctly parameters manipulating dialog-matching:

- Inbound HMR
- Outbound HMR

The new setting **out-of-dialog** (for the **msg-type** parameter) addresses these challenges by offering an intelligent more of dialog matching of messages for inbound and outbound HMR requests. This is a msg-type parameter, meaning that it becomes matching criteria for operations performed against a message. If you also specify methods (such as REGISTER) as matching criteria, then the rule is further limited to the designated method.

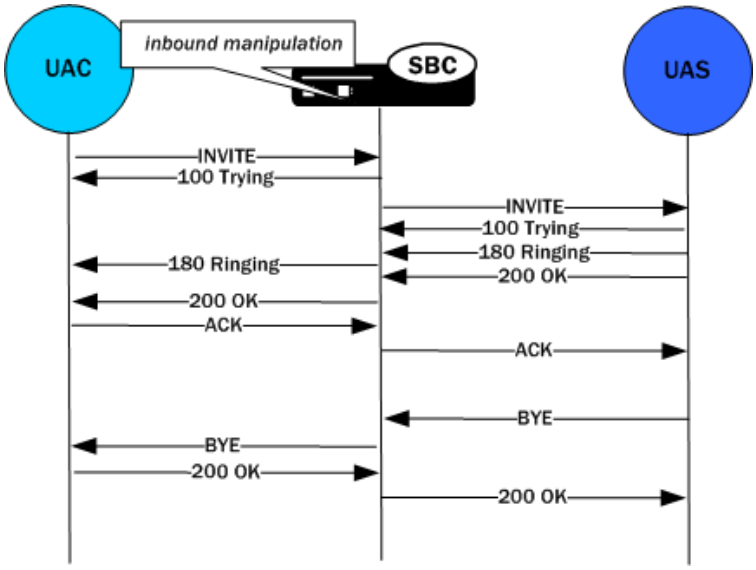
For both inbound and outbound manipulations, using the **out-of-dialog** setting means the message must be a request without a to-tag in order to perform the manipulation.

Inbound HMR Challenge

Because inbound manipulations take place before the message reaches the core of Oracle Communications Session Border Controller (SBC) SIP processing, the SIP proxy takes the manipulated header as directly received from the client. This can cause problems for requests leaving the SBC for the UAC because the dialog does not match the initial request sent.

The unmodified header must be cached because for any subsequent request (For example, a BYE originating from the terminator. See the following diagram.) the SBC might need to restore the original value, enabling the UAC to identify the message correctly as being part of the same dialog. For out-of-dialog requests (when the To, From, or Call-ID headers are modified) the original header is stored in the dialog when the **msg-type out-of-dialog** is used.

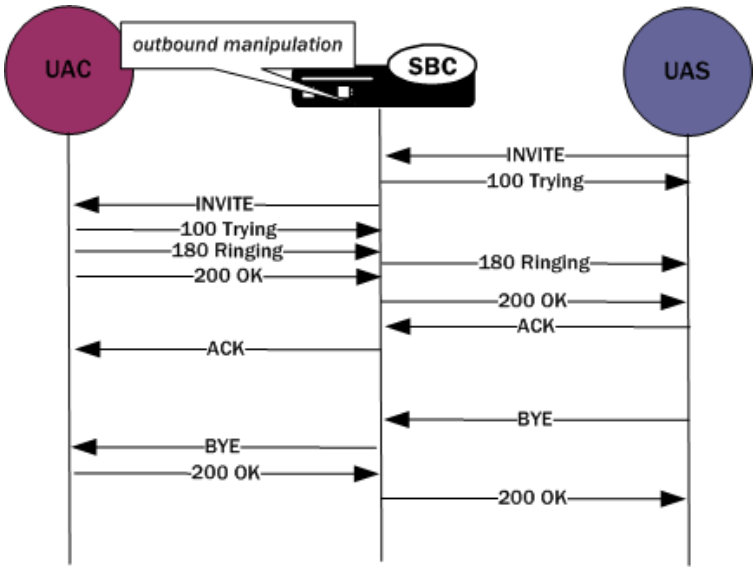
The SBC performs the restoration of original headers outside of SIP manipulations. There are no manipulation rules to configure for restore the header to their original context. The SBC recognizes that the headers are modified, and restores them to their original state prior to sending the message out. Restoration takes place prior to outbound manipulations so that any outbound manipulation can those headers after they are restored.



Outbound HMR Challenge

When you use the **out-of-dialog** setting for an outbound manipulation, the Oracle Communications Session Border Controller executes this specific SIP header rule only if the same SIP header rule was executed against the initial dialog-creating request.

For example, if the INVITE's To header was not manipulated, it would not be correct to manipulate the To header in the BYE request. To do so would render the UAC unable to properly match the dialog. And this also means that the outbound manipulation should be carried out against a To, From, or Call-ID header in the BYE request if it was manipulated in the INVITE.



Dialog-matching Header Manipulation Configuration

You using the **out-of-dialog** setting in the **msg-type** parameter, part of the SIP header rules configuration.

To enable dialog-matching header manipulation:

1. Access the **header-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)#
```

2. **msg-type**—Set this parameter to **out-of-dialog** to enable dialog-matching header manipulation.
3. Type **done** to save your work.

Built-In HMRs

In the course of HMR use, certain SIP manipulations have become commonly used. Oracle created a library of built-in SIP manipulations that you use exactly like the HMRs you create yourself. You apply the built-in HMRs as arguments to the **in-manipulationid** and **out-manipulationid** parameters for session agents, realms, and SIP interfaces. You can also apply them in HMR sets as a nested manipulation.

Built-in rules start with the prefix ACME_, so Oracle recommends you name your own rules in a different manner to avoid conflict.

You can view a list of built-in manipulations using the following ACLI command:

```
show built-in-sip-manipulation
```

ACME_NAT_TO_FROM_IP

When performed outbound, this rule changes:

- The To-URI hostname to the logical $TARGET_IP and port to $TARGET_PORT
- The From-URI to the logical $LOCAL_IP and port to be $LOCAL_PORT

> **Note**
>
> This built-in HMR only runs when the URI-host is an IP address, not when it's a domain name.

When applied in an inbound manner, the remote and local ports are reveresed.

Built-In Variables

To improve performance and reduce development complexity, the SBC contains built-in variables for common components of the SIP message. These reserved variables operate exactly like customer-defined variables. The recommended syntax is:

```
$<variable>.$0
```

For example:

```
$PAI_USER.$0
```

When you omit the `$0`, the resulting value is TRUE or FALSE, which can be useful to determine if there was no username in the PAI header or that no PAI header exists.

The values for the variables are obtained when they are resolved. For example, when the To-URI has been changed by a previous rule, the current rule gets the changed value (as would apply to $ORIGINAL). When the header or value does not exist in the SIP message, either an empty string is returned or, for Boolean uses, the value FALSE is returned.

The following table lists and describes the built-in variables.

Variable	Description
$ORIGINAL	Original value of element
$LOCAL_IP	IP address of the SIP interface on which the message was received for inbound manipulation or sent on for outbound manipulation.
$LOCAL_PORT	Port number of the SIP interface on which the message was received for inbound manipulation or sent on for outbound manipulation.
$REMOTE_IP	IP address the message was received from for inbound manipulation or sent to for outbound manipulation.
$REMOTE_PORT	Port number the message was received from for inbound manipulation or sent to for outbound manipulation.
$REMOTE_VIA_HOST	Host from the top Via header of the message.
$TRUNK_GROUP	Legacy reserved variable that can resolve to <TRUE/FALSE>.
$TRUNK_GROUP_CONTEXT	Legacy reserved variable that can resolve to <TRUE/FALSE>.
$REPLY_IP	IP address where the SIP message came from
$REPLY_PORT	Port number where the SIP message came from
$TARGET_IP	IP address where the SIP message is sent to
$TARGET_PORT	Port number where the SIP message is sent to
$MANIP_STRING	The manipulation string

Variable	Description
$MANIP_PATTERN	Use a regex pattern from the most specific matching session agent, realm, or SIP interface. Only this variable can be used in the match-value field. You cannot combine it with additional characters. This variable can be used in any rule you use a pattern-rule match value, including store action rules. You can also reference the stored values from those referenced in later rules. For example, you can create an allow list based on trunk From header uri-user parameter. Each session agent passes a different string on which to perform the allow list operation. Because the MANIP_PATTERN is dynamically decided at run-time every time the HMR executes for each message, it is possible no manipulation pattern will be found. In this scenario, it will use the default `\,+`. This default works most like `.+`. It is also possible a sub-group might be referenced that was not in the pattern chosen, in this scenario the variable resolves to empty/FALSE.
$CRLF	Search for carriage returns in new lines. Because you can search for these value and replace them, you also must be able to add them back in when necessary. Resolves to `\r\n` and is commonly used in MIME manipulation. If you are creating a new body, there might be a need for many CRLFs in the new-value parameter.
$TO_USER	URI username from To header without any user parameters.
$TO_PHONE	URI user of the To header as a phone number without any visual separators and with the leading + if it is present.
$TO_HOST	URI host portion from the To header.
$TO_PORT	URI port number from the To header. This is set to 5060 if it is not actually in the message.
$FROM_USER	URI username from the From header without any user parameters
$FROM_PHONE	URI user of the From header as a phone number without any visual separators and with the leading + if it is present
$FROM_HOST	URI host portion from the From header.
$FROM_PORT	URI port number from the From header. This is set to 5060 if it is not actually in the message.
$CONTACT_USER	URI username from the first instance of the Contact header without any user parameters.
$CONTACT_PHONE	URI user of the first instance of the Contact header as a phone number without any visual separators and with the leading + if it is present.
$CONTACT_HOST	URI host portion from the first instance of the Contact header
$CONTACT_PORT	URI port number from the first instance of the Contact header. This is set to 5060 if it is not actually in the message.

Variable	Description
$RURI_USER	URI username from the Request-URI header without any user parameters.
$RURI_PHONE	URI user of the Request-URI header as a phone number without any visual separators and with the leading + if it is present.
$RURI_HOST	URI host portion from the Request-URI header.
$RURI_PORT	URI port number from the Request-URI header. This is set to 5060 if it is not actually in the message.
$PAI_USER	URI username from the first instance of the P-Asserted-Identity header without any user parameters.
$PAI_PHONE	URI user of the first instance of the P-Asserted-Identity header as a phone number without any visual separators and with the leading + if it is present.
$PAI_HOST	URI host portion from the first instance of the P-Asserted-Identity header.
$PAI_PORT	URI port number from the first instance of the P-Asserted-Identity header. This is set to 5060 if it is not actually in the message.
$PPI_USER	URI username from the first instance of the P-Preferred-Identity header without any user parameters.
$PPI_PHONE	URI user of the first instance of the P-Preferred-Identity header as a phone number without any visual separators and with the leading + if it is present.
$PPI_HOST	URI host portion from the first instance of the P-Preferred-Identity header.
$PPI_PORT	URI port number from the first instance of the P-Preferred-Identity header. This is set to 5060 if it is not actually in the message.
$PCPID_USER	URI username from the P-Called-Party-ID header without any user parameters.
$PCPID_PHONE	URI user of theP-Called-Party-ID header as a phone number without any visual separators and with the leading + if it is present.
$PCPID_HOST	URI host portion from theP-Called-Party-ID header.
$PCPID_PORT	URI port number from the P-Called-Party-ID header. This is set to 5060 if it is not actually in the message.
$CALL_ID	Resolves to the Call-ID of the current SIP message; is a commonly stored rule.
$TIMESTAMP_UTC	Gets the current time from the SBC's system clock in RFC 3339 format: YYYY-MM-DDTHH:MM:SS.PPPZ The PPP is partial seconds and the time is based on UTC. For example: 2012-01-01 T22:00:09.123Z
$T_GROUP	Trunk group
$T_CONTEXT	Trunk group context

Variable	Description
$M_STRING	A boolean that is true if the manipulation-string exists, false if not. The variable $M_STRING.$0 contains the manipulation-string.

The $TARGET_IP and $REMOTE_IP Variables

The $TARGET_IP variable is always the next hop IP address for a SIP request. This variable is set only for requests.

The $REMOTE_IP variable changes depending on the state of the message. For outbound manipulations, the $REMOTE_IP variable is the destination IP address or next hop IP address; that is, the $REMOTE_IP and $TARGET_IP are the same for outbound manipulations. For inbound manipulations of a SIP request, the $REMOTE_IP variable is the reply IP address; that is, the $REMOTE_IP and $REPLY_IP are the same. For all other inbound manipulations, the $REMOTE_IP is set using the From header.

Built-In SIP Manipulation Configuration

When you want to enable this feature for a realm, session agent, or SIP interface, you configure the **in-manipulationid or out-manipulationid** parameters with the rule.

The sample here shows this feature being applied to a session agent, but the realm and SIP interface configurations also have the same parameter you use to set up the feature.

To use built-in SIP manipulations:

1. Access the **session-agent** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# session-agent
ORACLE(session-agent)
```

2. **out-manipulationid**—Enter name of the built-in rule you want to use.

> **Note**
>
> All built-in rules start with `ACME_`.

3. Save your work.

Unique Regex Patterns Per Peer and Trunk

The built-in variable $MANIP_PATTERN reduces the complexity of writing HMRs for multiple peers and trunks.

Similar to the reserved variable $MANIP_STRING, the variable $MANIP_PATTERN uses a regex pattern from the most-specific matching session-agent, realm, or sip-interface. Within these configuration objects, the "manipulation-pattern" attribute allows you to set a unique regex pattern. Only one regex pattern can be specified in the configuration attribute, and only the :variable $MANIP_PATTERN can appear in the match-value field (for example; the "$MANIP_PATTERN" cannot be combined with additional characters in the match-value).

This feature enables service providers to configure one or a few common global HMRs, while having a unique regex pattern for each SIP trunk or peer. It reduces the number of sip-manipulation sets that need to be provisioned, reducing provisioning work and system memory usage.

The $MANIP_PATTERN can be used in any rule you can use a pattern-rule match-value in, including store action rules. You can also reference the stored values from those referenced in later rules for example: using the $RuleName for the Boolean TRUE or FALSE, or $RuleName.$0 for the whole matching string). For example, you can create an allow list based on the trunk From header uri-user parameter. Each session-agent then passes a different string on which to perform the allow list operation. The following code block shows a configuration example:

```
sip-manipulation
    name                            sipTrunkallowlist
    ...
    header-rule
        name                        allowlistOnFrom
        header-name                 From
        action                      manipulate
        comparison-type             case-sensitive
        msg-type                    out-of-dialog
        methods                     INVITE
        match-value
        new-value
        element-rule
            name                    checkFromUriUser
            parameter-name
            type                    uri-user
            action                  store
            match-val-type          any
            comparison-type         pattern-rule
            match-value             $MANIP_PATTERN
            new-value
        element-rule
            name                    rejectIfNoMatch
            parameter-name
            type                    uri-user
            action                  reject
            match-val-type          any
            comparison-type         boolean
            match-value             !$allowlistOnFrom.$checkFromUriUser
            new-value               403:Forbidden

session-agent
    hostname                    172.16.50.101
    ip-address                  172.16.50.101
    port                        5060
    realm-id                    peer1-core
    ...
    manipulation-string
    manipulation-pattern        ^78132841([0-4][0-9])$

session-agent
    hostname                    172.16.50.102
    ip-address                  172.16.50.102
```

```
    port                        5060
    realm-id                    peer2-core
    ...
    manipulation-string
    manipulation-pattern        ^78132841([5-9][0-9])$
```

Rejecting SIP Requests

SIP requests can be rejected using HMRs.

To simplify rejecting SIP requests with HMRs, the SBC supports the reject action in any rule type. This rejects SIP requests if the conditions within the rule (match-value, msg-type, etc.) are true. When a SIP message is rejected, the SBC increments the counter called "Rejected Messages," which can be displayed in the ACLI with the `show sip transport` command. SIP responses cannot be rejected but the counter is still incremented.

A new MIB object in the ap-smgmt.mib for SNMP GET is available to obtain the counter value. The SBC can send an SNMP trap when the counter exceeds a configured threshold in a configured time window. The threshold is set by new “reject-message-threshold” and “reject-message-window” config attributes in session-router config.

When rejecting a matching SIP Request, a response-code and reason-phrase can be specified. In the rule configured with the "reject" action, enter the syntax `status-code[:reason-phrase]` in the new-value field. For example `401:Denied` in the new-value of a reject action rule will cause the SD to reject the SIP Request with a 401 response and "Denied" as the reason-phrase.

```
sip-manipulation
    name                        rejectINV
    description
    header-rule
        name                    from508
        header-name             from
        action                  manipulate
        comparison-type         case-sensitive
        msg-type                any
        methods                 INVITE
        match-value
        new-value
        element-rule
            name                fromUser
            parameter-name
            type                uri-phone-number-only
            action              reject
            match-val-type      any
            comparison-type     case-sensitive
            match-value         5085551212
            new-value           401:Denied
```

> **Note**
>
> When a SIP request matches a rule with a reject action, the rejection is immediate and later rules aren't executed.

> **Note**
>
> The reject action should not respond with a 200 OK. Instead, use a response code in the 400, 500, or 600 range.

HMR Information in Logs

You can apply an action type called **log** to all manipulation rules. When you use this action type and a condition matching the manipulation rule arises, the SBC logs information about the current message to a separate log file. This log files will be located on the same core in which the SIP manipulation occurred. On the core where sipt runs, a logfile called matched.log will appear when this action type is executed.

The matched.log file contains a timestamp, received and sent SBC network interface, sent or received IP address:port information, and the peer IP address:port information. It also specifies the rule that triggered the log action in this syntax: `rule-type[rule name]`. The request URI, Contact header, To Header, and From header are also present.

```
--------------------------------------
Apr 17 14:17:54.526 On [0:0]192.168.1.84:5060 sent to 192.168.1.60:5060
element-rule[checkRURIPort]
INVITE sip:service@192.168.1.84:5060 SIP/2.0
From: sipp <sip:+2125551212@192.168.1.60:5060>;tag=3035SIPpTag001
To: sut <sip:service@192.168.1.84>
Contact: sip:sipp@192.168.1.60:5060
```

Using Regular Expressions

Regular expressions (regex) are patterns that describe character combinations in text. Regex provides a concise and flexible means to match strings of text, such as particular characters, words, or patterns of characters. SIP messages are treated as sets of substrings on which regex patterns rules are executed. With regex you can create strings to match other string values and use groupings in order to create stored values on which to operate.

> **Note**
>
> An understanding of regex is required for successful HMRs. Refer to *Mastering Regular Expressions* from O'Reily Media for more information.

Oracle's SBC supports the standardized regular expression format called Portable Operating System Interface (POSIX) Extended Regular Expressions. The SBC regex engine is a traditional regex-directed (NFA) type.

Example of HMR with Regex

The following HMR removes a P-Associated-URI from an response to a REGISTER request. The regex expression `^<tel:` lets you specify the removal only if it is a tel-URI.

```
sip-manipulation
    name                            rem_telPAU
    description
```

```
header-rule
    name                    modPAU
    header-name             P-Associated-URI
    action                  delete
    comparison-type         pattern-rule
    match-value             ^<tel:
    msg-type                reply
    new-value
    methods                 REGISTER
```

Regex Characters

Regular expressions are used to search for patterns of text using one or more of the following devices:

Character Type	Example	Description
Literal text	`foobar`	With the exception of a small number of characters that have a special meaning in a regex, text matches itself.
Special wildcard characters	`\d`	Known as metacharacters or metasequences, these match or exclude specific types of text, such as any number.
Character classes	`[1-5]`	When a suitable metacharacter or metasequence doesn't exist, you can create your own definition to match or exclude specified characters.
Quantifiers	+ or ?	These specify how many times you want the preceding expression to match or whether it's optional.
Capturing groups and backreferences	`(foobar)` or `\1`	These specify parts of the regex that you want remembered, either to find a similar match later on, or to preserve the value in a find and replace operation.
Boundaries and anchors	^ or $	These specify where the match should be made, for example at the beginning of a line or word.
Alternation	|	This specifies alternatives.

By default, regular expressions are case-sensitive, so `A` and `a` are treated as different characters. As long as what you're looking for fits a regular pattern, a regex can be created to find it.

Literal (Ordinary)

Many of the characters you can type on your keyboard are literal, ordinary characters; they present their actual value in the pattern. For example, the regex pattern `sip`, is a pattern of all literal characters, that will be matched from left to right, at each position in the input string, until a match is found. Given an input string of `<sip:me@here.com>`, the regex pattern `sip` will successfully match the `sip`, starting at the position of the `s` and ending at the position of the `p`. But the same regex will also match `sip` in `<sips:me@here.com>` and

`tel:12345;isip=192.168.0.3` because an `s` followed by an `i` followed by a `p` exists in both of those as well.

Special (Metacharacters)

Some characters have special meaning. They instruct the regex function (or engine which interprets the expressions) to treat the characters in designated ways. The following table outlines these special characters or metacharacters.

Character	Name	Description
`.`	dot	Matches any one character, including a space; it will match one character, but there must be one character to match. Matches a literal dot when bracketed or placed next to a backslash: `[.]` or `\.`.
*	star/asterisk	Matches one or more preceding character (0, 1, or any number), bracketed carrier class, or group in parentheses. Used for quantification. Typically used with a dot in the format `.*` to indicate that a match for any character, 0 or more times. Matches a iteral asterisk when bracketed: `[*]`.
+	plus	Matches one or more of the preceding character, bracketed carrier class, or group in parentheses. Used for quantification. Matches a literal plus sign when bracketed: `[+]`.
|	bar/vertical bar/pipe	Matches anything to the left or to the right; the bar separates the alternatives. Both sides are not always tried; if the left does not match, only then is the right attempted. Used for alternation.
{	left brace	Begins an interval range, ended with } (right brace) to match; identifies how many times the previous single character or group in parentheses must repeat. Interval ranges are entered as minimum and maximums `{minimum,maximum}` where the character or group must appear a minimum number of times up to the maximum. You can also use interval ranges to set magnitude, or exactly the number of times a character must appear; you can set this, for example, as the minimum value without the maximum `{minimum,}`.
?	question mark	Signifies that the preceding character or group in parentheses is optional; the character or group can appear not at all or one time.
^	caret	Acts as an anchor to represent the beginning of a string.
$	dollar sign	Acts as an anchor to represent the end of a string.
[	left bracket	Acts as the start of a bracketed character class, ended with the] (right bracket). A character class is a list of character options; one and only one of the characters in the bracketed class must appear for a match. A - (hyphen) in between two characters enclosed by brackets designates a range; for example `[a-z]` is the character range of the lower case twenty-six letters of the alphabet. Note that the] (right bracket) ends a bracketed character class unless it sits directly next to the [(left bracket) or the ^ (caret); in those two cases, it is the literal character.
(	left parenthesis	Creates a grouping when used with the) (right parenthesis). Groupings have two functions: Separate pattern strings so that a whole string can have special characters within it as if it were a single character. They allow the designated pattern to be stored and referenced later (so that other operations can be performed on it).

Regex Tips

- Limit use of wildcards asterisk * and plus sign +.
- A character class enclosed by brackets [] is not a choice of one or more characters but rather a choice of one and only one character in the set.
- The range 0-1000 is not the same as the range 0000-1000.
- Spaces are legal characters and will be interpreted like any other character.

Matching New Lines

In the regular expression library, the dot . character does not match new lines or carriage returns. Conversely, the not-dot does match new lines and carriage returns. This provides a safety mechanism preventing egregious backtracking of the entire SIP message body when there are no matches. The SBC reduces backtracking to a single line within the body.

Escaped Characters

SIP HMR's support for escaped characters allows for searches for values you would be unable to enter yourself. Because they are necessary to MIME manipulation, support for escaped characters includes:

Syntax	Description
\s	Whitespace
\S	Non-whitespace
\d	Digits
\D	Non-digits
\R	Any \r, \n, or \r\n
\w	Word
\W	Non-word
\A	Beginning of buffer
\Z	End of buffer
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab

Building Expressions with Parentheses

You can use parentheses ( ) when you use HMR to support order of operations and to simplify header manipulation rules that might otherwise prove complex. This means that expressions such as `(sip + urp) - (u + rp)` can now be evaluated to `sip`. Previously, the same expression would have evaluated to `sipurprp`. In addition, you previously would have been required to create several different manipulation rules to perform the same expression.

Boolean Operators

The following Boolean operators are supported:

- `&`, meaning AND.
- `|`, meaning OR.
- `!`, meaning NOT.

You can only use Boolean operators when the **comparison type** is pattern-rule and you are evaluating stored matches. The SBC evaluates these Boolean expressions from left to right, and does not support any grouping mechanisms that might change the order of evaluation. For example, the SBC evaluates the expression `A & B | C` (where A=true, B=false, and C=true) as follows: A & B = false; false | true = true.

Equality Operators

You can use equality operators in conjunction with string operators. You can also use equality operators with:

- Boolean operators, as in this example: `($rule1.$0 == $rule2.$1) & $rule3`.
- The `!`, `&`, and `|` operators.
- Variables and constant strings.

You can group them in parentheses for precedence.

Equality operators always evaluate to either true or false.

Equality Operator Symbol	Short Description	Detailed Information
==	String case sensitive equality operator	Performs a character-by-character, case-sensitive string comparison on both the left side and the right side of the operator.
~=	String case insensitive equality operator	Performs a character-by-character, case-insensitive string comparison on both the left side and the right side of the operator.
!=	String case sensitive inequality operator	Performs a character-by-character, case-sensitive string comparison on both the left side and the right side of the operator, returning true if the left side is not equal to the right side.
<=	Less than or equal to operator	Performs a string-to-integer conversion. If the string-to-integer comparison fails, the value is treated as 0. After the conversion, the operator will compare the two values and return true only if the left side is less than or equal to the right side of the operator.
>=	Greater than or equal to operator	Performs a string-to-integer conversion. If the string-to-integer comparison fails, the value is treated as 0. After the conversion, the operator will compare the two values and return true only if the left side is greater than or equal to the right side of the operator.
<	Less than operator	Performs a string-to-integer conversion. If the string-to-integer conversion fails, the value is treated as 0. After the conversion, the operator will compare the two values and return true only if the left side is less than the right side of the operator.
>	Greater than operator	Performs a string-to-integer conversion. If the string-to-integer conversion fails, the value is treated as 0. After the conversion, the operator will compare the two values and return true only if the left side is greater than the right side of the operator.

Normalizing EBNF ExpressionString Grammar

The expression parser grammar implies that any expression string can have boolean and string manipulation operators in the same expression. While technically this is possible, the expression parser prevents it.

Because all boolean expressions evaluate to the string value TRUE or FALSE and since all manipulation are string manipulations, the result of a boolean expression returns the value TRUE or FALSE. The ExpressionString class interprets this as an actual TRUE or FALSE value. For this reason, boolean operators are not mixed with string manipulation operators (which is true with most programming languages).

The expression string grammar also indicates that it is possible to nest self-references and rule names indefinitely. For HMR, this is not allowed. A self-reference can only exist by itself, and a terminal index can only come at the end of a rule reference.

Storing Regex Patterns

Any HMR with a pattern-rule comparison type can store a regex pattern's matches for later use. In many cases you don't have to create store rules before manipulation rules. Data is only stored for items that later rules actually reference.

For example, if a later rule never references a header rule's stored value, but only its element rules, then the header rule itself doesn't store anything. Alternatively, you could delete a header or field, but still use its stored value later without having to create a separate store rule for it. In general, fewer rules improve SBC performance.

Performance Considerations

The regex engine consumes as much of the input string as it can before it backtracks or gives up trying, which is called greediness. Greediness can introduce errors in regex patterns and has an effect on performance. There is usually a trade-off of efficiency versus exactness - you should choose how exacting you need to be. Keep the following in mind in order to lessen the effect:

- Poorly constructed regex patterns can effect the performance of regex matching for long strings
- Search on the smallest input string possible, perform a regex search in element rules for the specific header component type you want to match for
- Test the regex pattern against long strings which do not match to evaluate the effect on performance.
- Test a regex with a wildcard in between characters against an input string with those characters repeated in different spots to evaluate performance
- If the input string format is fairly fixed and well-known, be explicit in the regex rather than using wildcards
- If the regex pattern is trying to capture everything before a specific character, use the negation of the character for the wildcard character. Note that this is true most times, except when there is an anchor at the end.
- Use beginning-line and ending-line anchors whenever possible if you want to only match if the pattern begins or ends as such.
- A dot . means any character, including whitespace. A wild-carded dot, such as .* or .+, will capture/match everything until the end of line, and then it will backtrack if there are more

characters after the wildcard that need to be matched. If you don't need to capture the things before the characters after the wildcard, don't use the wildcard.

Additional References

To learn more about regex, you can visit the following Web site, which has information and tutorials that can help to get you started:http://www.regular-expressions.info/.

HMR Configuration

To configure SIP header and parameter manipulation, first create a SIP header manipulation ruleset. Then create the header manipulation rules and optional header element rules for that ruleset to contain. Then configure a session agent or a SIP interface to use the SIP header and parameter manipulation ruleset in the inbound and outbound directions.

Testing Pattern Rules

Use **test-pattern-rule** to test the effect of your regex patterns.

1. Access the **test-pattern-rule** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# test-pattern-rule
ORACLE(test-pattern-rule)#
```

2. **expression**—Enter the regular expression to test.
3. **string**—Enter the string against which you want to compare the regular expression.
4. **show**—View the test pattern, the string, and the matches.

```
ORACLE(test-pattern-rule)# expression ".*(;tgid=(.+)).*"
expression made 0 matches against string
ORACLE(test-pattern-rule)# string "sip:+17024260002@KCMGGWC;user=phone SIP/
2.0;tgid=Trunk1"
expression made 3 matches against string
ORACLE(test-pattern-rule)# show
Pattern Rule:
        Expression : .*(;tgid=(.+)).*
        String     : sip:+17024260002@KCMGGWC;user=phone SIP/2.0;tgid=Trunk1
        Matched    : TRUE
        Matches:
$0 sip:+17024260002@KCMGGWC;user=phone SIP/2.0;tgid=Trunk1
$1 ;tgid=Trunk1
$2 Trunk1

ORACLE(test-pattern-rule)#
```

Creating Header Manipulation Rulesets

First create a header rule and then create element rules within that header rule.

1. Access the **sip-manipulation** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)#
```

2. **name**—Enter the name you want to use for this ruleset.
3. **split-headers**—Enter a comma-separated list of headers to be split and treated as separate headers. The splitting of headers occurs prior to the execution of any manipulation rules.

```
ORACLE(sip-manipulation)# split-headers Diversion,Route,Via
```

4. **join-headers**—Enter a comma-separated list of headers to be joined into a single comma-separated header. The joining of headers occurs after execution of any manipulation rules.

```
ORACLE(sip-manipulation)# join-headers Diversion,Route,Via
```

5. Access the **header-rules** configuration element.

```
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)#
```

6. **name**—Enter a unique name for this rule.
7. **header-name**—Enter the name of the header to which this rule applies.

   The name entered here is a case-insensitive string that must match a header name. Create a rule using the long form of the header name and a rule using the compact form of the header name.

   > **Note**
   >
   > The Request-URI header is identified as request-uri.

8. **action**—Enter the action you want applied to the header specified in the name parameter.

   The default value is none. Valid options are:

   - add—Add a new header, if that header does not already exist.
   - delete—Delete the header, if it exists.
   - manipulate—Elements of this header will be manipulated according to the element rules configured.
   - store—Store the header.
   - none—No action to be taken.

9. **match-value**—Enter the value to be matched (only an exact match is supported) with a header value.

   The action specified is only performed if the header value matches.

10. **msg-type**—Enter the message type to which this header rule applies.

    The default value is any. Valid options are:

- any—Both Requests and Reply messages
- request—Request messages only
- reply—Reply messages only

11. **methods**—Enter the SIP method names to which you want to apply this header rule. If entering multiple method names, separate them with commas. For example:

    ```
    INVITE,ACK,BYE
    ```

    Leaving the method field empty applies the header-rule to all methods.

12. Access the **element-rules** configuration element.

    The **element-rules** configuration element defines the element rules, which are executed on those elements of the header specified by the header rule.

    ```
    ORACLE(sip-header-rules)# element-rules
    ORACLE(sip-element-rules)#
    ```

    a. **name**—Enter the name of the element to which this rule applies.

    > **Note**
    >
    > The name parameter usage depends on the element type you enter in step 6. For uri-param, uri-user-param, and header-param it is the parameter name to be added, replaced, or deleted. For all other types, it serves to identify the element rule and any name can be used.

    b. **type**—Enter the type of element on which to perform the action.

    The default value is none. Valid options are:

    - header-value—Enter value of the header.
    - header-param-name—Header parameter name.
    - header-param—Parameter portion of the header.
    - uri-display—Display of the SIP URI.
    - uri-user—User portion of the SIP URI.
    - uri-host—Host portion of the SIP URI.
    - uri-port—Port number portion of the SIP URI.
    - uri-param-name—Name of the SIP URI param.
    - uri-param—Parameter included in the SIP URI.
    - uri-header-name—SIP URI header name
    - uri-header—Header included in a request constructed from the URI.
    - uri-user-param—User parameter of the SIP URI.

    c. **action**—Enter the action you want applied to the element specified in the name parameter, if there is a match value.

    The default value is none. Valid options are:

    - none—No action is taken.

- add—Add a new element, if it does not already exist.
- store—Store the elements.
- replace—Replace the elements
- delete-element—Delete the specified element if it exists.
- delete-header—Delete the specified header, if it exists.

d. **match-val-type**—Enter the type of value that needs to be matched to the match-field entry for the action to be performed.

The default value is ANY. Valid options are:

- IP—Element value in the SIP message must be a valid IP address to be compared to the match-value field entry. If the match-value field is empty, any valid IP address is considered a match. If the element value is not a valid IP address, it is not considered a match.
- FQDN—Element value in the SIP message must be a valid FQDN to be compared to the match-value field entry. If the match-value field is empty, any valid FQDN is considered a match. If the element value is not a valid FQDN, it is not considered a match.
- ANY—Element value in the SIP message is compared with the match-value field entry. If the match-value field is empty, all values are considered a match.

e. **match-value**—Enter the value you want to match against the element value for an action to be performed.

f. **new-value**—Enter the value for a new element or to replace a value for an existing element. You can enter an expression that includes a combination of absolute values, pre-defined parameters, and operators

> **Note**
>
> Absolute values, with which you can use double quotes for clarity. You must escape all double quotes and back slashes that are part of an absolute value, and enclose the absolute value in double quotes.

Examples of entries for the new-value field.

```
sip:"+$TRUNK_GROUP+".$TRUNK_GROUP_CONTEXT
$ORIGINAL+acme
$ORIGINAL+"my name is john"
$ORIGINAL+"my name is \"john\""
$ORIGINAL-^781+^617
```

g. Type **done** and **exit** to save the rule and return to the header-rules configuration element.

13. Type **done** and **exit** to save the rule and return to the **sip-manipulation** configuration element.

Configuring SIP Header Manipulation Rules

To configure dynamic SIP header manipulation rules:

1. Access the **header-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)#
```

2. **name**—Enter the unique identifier for this SIP HMR.

   This configuration element has no default value.

3. **header-name**—Enter the name of the header on which to operate.

   This configuration element has no default value.
   Set this parameter to **@status-line** to prevent undesired matches with header having the name status-code.

4. **msg-type**—Specify the type of message to which this SIP HMR will be applied.

   The default value is **any**. Valid values are:

   - any
   - request
   - reply

5. **methods**—Enter the method type on which to operate.

   When you do not set the method, the SBC applies the rule across all SIP methods. Valid values are:

   - INVITE
   - ACK
   - CANCEL

6. **comparison-type**—Enter the way in which the SBC will process match rules against SIP headers.

   The default is **case-sensitive**. The valid values are:

   - boolean
   - refer-case-sensitive
   - pattern-rule
   - case-sensitive
   - case-insensitive

7. **action**—Enter the action to perform on the SIP header.

   The default value is **none**. The valid values are:

   - add
   - delete
   - manipulate
   - store
   - none

> **Note**
>
> Remember that you should enter rules with the action type store before you enter rules with other types of actions.

If the action type is set to store, the SBC treats the match value as a regular expression. As a default, the regular expression used for the match value is `.+` (which indicates a match value of at least one character), unless you set a more specific regular expression match value.

8. **match-value**—Enter the value to match against the header value.

   The SBC matches these against the entire SIP header value. This is where you can enter values to match using regular expressions. Your entries can contain Boolean operators. When you configure HMR (using SIP manipulation rules, elements rules, etc.), you can use escape characters to support escaping Boolean and string manipulation operators.

9. **new-value**—When the action parameter is set to add or to manipulate, enter the new value that you want to substitute for the entire header value.

   This is where you can set stored regular expression values for the SBC to use when it adds or manipulates SIP headers.

Configuring SIP Header Manipulation Element Rules

Element rules are a subset of the SIP header manipulation rules and are applied at the element type level rather than at the entire header value.

To configure dynamic SIP header manipulation rules:

1. Access the **element-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# element-rules
ORACLE(sip-element-rules)#
```

2. **name**—Enter the unique identifier for this element rule.

   There is no default value.

3. **parameter-name**—Enter the SIP header parameter or element on which to operate.

   There is no default value.

4. **type**—Specify the type of parameter to which this element rule will be applied.

   The default value is **none**. The valid values are:

   - header-value
   - header-param-name
   - header-param
   - uri-display
   - uri-user
   - uri-user-param

- uri-host
- uri-port
- uri-param-name
- uri-param
- uri-header-name
- uri-header

To configure HMR so that only the status-line is affected, set comparison-type to one of the following:

- status-code—Designates the status code of the response line; accepts any string, but during the manipulation process only recognizes the range from 1 to 699.
- reason-phrase—Designates the reason of the response line; accepts any string.

5. **match-val-type**—Enter the value type that you want to match when this rule is applied.

   The default value is **ANY**. Valid values are:

   - IP
   - FQDN
   - ANY

6. **comparison-type**—Enter the way that you want SIP headers to be compared from one of the available.

   This choice dictates how the SBC processes the match rules against the SIP header parameter/element. The default is **case-sensitive**.

   - boolean
   - refer-case-sensitive
   - refer-case-insensitive
   - pattern-rule

7. **action**—Enter the action that you want this rule to perform on the SIP header parameter/element.

   The default is **none**. The valid rules are:

   - add
   - replace
   - delete-element
   - delete-header
   - store
   - none

   Remember that you should enter rules with the action type store before you enter rules with other types of actions.

   When you set the action type to store, the SBC always treats the match value you enter as a regular expression. As a default, the regular expression is uses for the match value is `.+` (which indicates a match value of at least one character), unless you set a more specific regular expression match value.

8. **match-value**—Enter the value to match against the header value in SIP packets.

The SBC matches these against the value of the parameter/element. This is where you can enter values to match using regular expression values, or stored pattern matches. Your entries can contain Boolean operators.

9. **new-value**—When the action parameter is set to add or to manipulate, enter the new value that you want to substitute for the entire header value.

   This is where you can set stored regular expression values for the SBC to use when it adds or manipulates parameters/elements.

Status-Line Manipulation and Value Matching

The SBC's HMR feature has been enhanced to support the ability to change the status code or reason phrase in SIP responses. This addition—the ability to edit status-lines in responses—builds on HMR's existing ability to edit response headers or the Request-URI in a request.

This section shows you how to configure SIP HMR when you want the SBC to drop a 183 Session Progress response when it does not have SDP, though flexibility is built into this feature so that you can use it to achieve other ends. In addition, you can now set the SIP manipulation's **match-value** parameter with Boolean parameters (AND or OR).

Set the Header Name

Set the header-name to **@status-line** to modify the status code or reason phrase in SIP responses.

1. Access the **header-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)#
```

2. **header-name**—Enter **@status-line**.

```
ORACLE(sip-header-rules)# header-name @status-line
ORACLE(sip-header-rules)#
```

Set the Element Type

In the **element-rules** configuration element, set the **type** parameter to either **status-code** or **reason-phrase**.

- **status-code**—Designates the status code of the response line. Accepts any string, but during the manipulation process only recognizes the range from 1 to 699.
- **reason-phrase**—Designates the reason of the response line. Accepts any string.

> **Note**
>
> Like other rule types, the Oracle Communications Session Border Controller matches against the value for these using case-sensitive, case-insensitive, or pattern-rule matching (set in the **comparison-type** parameter for the element rule).

1. Access the **element-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# element-rules
ORACLE(sip-element-rules)#
```

2. **type**—Enter either **status-code** or **reason-phrase**.

```
ORACLE(sip-element-rules)# type status-code
```

The SBC uses the value of **comparison-type** to determine matching.

Set the Match Value

Set the match value in either the **header-rules** configuration element or the **element-rules** configuration element

Set the Header Rules Match Value

Set a match value in the **header-rules** configuration element.

1. Access the **header-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)#
```

2. **match-value**—Enter the value to match against the header value.

   The Oracle Communications Session Border Controller matches these against the entire SIP header value. This is where you can enter values to match using regular expression values; your entries can contain Boolean operators.

Set the Element Rules Match Value

Set a match value in the **element-rules** configuration element.

1. Access the **element-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# element-rules
ORACLE(sip-element-rules)#
```

2. **match-value**—Enter the value to match against the header value.

   The Oracle Communications Session Border Controller matches these against the entire SIP header value. This is where you can enter values to match using regular expression values; your entries can contain Boolean operators.

Set the Response Code Block

Enable SIP response blocking to keep the Oracle Communications Session Border Controller from sending the designated response.

> **Note**
>
> This example sets the dropResponse option to 699, where 699 is an arbitrary code used to later match the HMR.

1. Access the **sip-interface** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-interface
ORACLE(sip-interface)#
```

2. Select the **sip-interface** object to edit.

```
ORACLE(sip-interface)# select
<RealmID>:
1: realm01 172.172.30.31:5060

selection: 1
ORACLE(sip-interface)#
```

3. **options**—Enter **options +dropResponse=<response code>** where <response code> is the code(s) or range(s) to block. Separate multiple entries with a colon.

```
ORACLE(sip-interface)# options +dropResponse=699
```

> **Warning**
>
> Typing the option without the plus sign will overwrite previously configured options. To append the options to this configuration's options list, prepend the option with a plus sign.

4. Save and activate your configuration.

Configuring SIP HMR Sets

To enable HMR sets, set the **action** configuration element to **sip-manip**.

1. Access the **element-rules** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# element-rules
ORACLE(sip-element-rules)#
```

2. **action**—Enter sip-manip value to enable use this rule for a SIP HMR set. This value then invoke the rule identified in the new-value parameter.
3. **new-value**—Enter the name of the manipulation rule you want invoked for the set.
4. Type **done** to save your configuration.
5. Run **verify-config** to detect invalid or circular references.
6. Save and activate your configuration.

Configuring a Session Agent

Configure a session agent to use a SIP header manipulation ruleset.

1. Access the **session-agent** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# session-agent
ORACLE(session-agent)
```

2. **in-manipulationid**—Enter the name of the SIP header manipulation ruleset you want to apply to inbound SIP packets.

```
ORACLE(session-agent)# in-manipulationid route-stripper
```

3. **out-manipulationid**—Enter the name of the SIP header manipulation ruleset you want to apply to outbound SIP packets.

```
ORACLE(session-agent)# out-manipulationid route-stripper
```

4. Type **done** to save your configuration.

Configuring a SIP Interface

Configure a interface to use a SIP header manipulation ruleset.

1. Access the **sip-interface** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-interface
ORACLE(sip-interface)#
```

2. **in-manipulationid**—Enter the name of the SIP header manipulation ruleset to apply to SIP packets in the ingress direction.

```
ORACLE(sip-interface)# in-manipulationid topology-hiding
```

3. **out-manipulationid**—Enter the name of the SIP header manipulation ruleset to apply to SIP packets in the egress direction.

```
ORACLE(sip-interface)# out-manipulationid topology-hiding
```

4. Type **done** to save your configuration.

Example 1 Stripping All Route Headers

This example explains how to strip all route headers from a SIP packet. First, you create a header manipulation ruleset, in the example it is called route-stripper. Then you configure the list of header manipulation rules you need to strip route headers. In this case, you only need one rule named Route (to match the Route header name) with the action set to Delete.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# name route-stripper
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# name Route
ORACLE(sip-header-rules)# action Delete
ORACLE(sip-header-rules)# done
header-rule
        name                            Route
        action                          delete
        match-value
        msg-type                        any
ORACLE(sip-header-rules)# ex
ORACLE(sip-manipulation)# done
sip-manipulation
        name                            route-stripper
        header-rule
                name                            Route
                action                          delete
                match-value
                msg-type                        any
```

Example 2 Stripping an Existing Parameter and Adding a New One

This example explains how to strip the user parameter from the Contact header URI and add the acme parameter with value as LOCAL IP, only for requests. First you create a header manipulation ruleset, in the example it is called param-stripper1. You then configure a list of header rules you need. In this case, you only need one rule named Contact (to match the Contact header name), with action set to manipulate (indicating the elements of this header would be manipulated). Next, you configure a list of element rules for the Contact header rule.

In this case you configure two element rules; one to strip the uri parameter user (the rule name user matches the param name user) and the other to add the uri parameter acme (the rule name acme matches the param name acme).

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)# name param-stripper1
ORACLE(sip-manipulation)# header-rules
ORACLE(sip-header-rules)# name Contact
ORACLE(sip-header-rules)# action manipulate
ORACLE(sip-header-rules)# msg-type request
ORACLE(sip-header-rules)# element-rules
ORACLE(sip-element-rules)# name user
ORACLE(sip-element-rules)# type uri-param
```

```
ORACLE(sip-element-rules)# action delete-element
ORACLE(sip-element-rules)# done
element-rule
        name                            user
        type                            uri-param
        action                          delete-element
        match-val-type                  any
        match-value
        new-value
ORACLE(sip-element-rules)# name acme
ORACLE(sip-element-rules)# action add
ORACLE(sip-element-rules)# type uri-param
ORACLE(sip-element-rules)# new-value "$LOCAL_IP"
ORACLE(sip-element-rules)# done
element-rule
        name                            acme
        type                            uri-param
        action                          add
        match-val-type                  any
        match-value
        new-value                       "$LOCAL_IP"
ORACLE(sip-element-rules)# ex
ORACLE(sip-header-rules)# done
header-rule
        name                            Contact
        action                          manipulate
        match-value
        msg-type                        request
        element-rule
                name                            user
                type                            uri-param
                action                          delete-element
                match-val-type                  any
                match-value
                new-value
        element-rule
                name                            acme
                type                            uri-param
                action                          add
                match-val-type                  any
                match-value
                new-value                       "$LOCAL_IP"
ORACLE(sip-header-rules)# ex
ORACLE(sip-manipulation)# done
sip-manipulation
        name                            param-stripper1
        header-rule
                name                            Contact
                action                          manipulate
                match-value
                msg-type                        request
                element-rule
                        name                            user
                        type                            uri-param
                        action                          delete-element
                        match-val-type                  any
```

```
                        match-value
                        new-value
element-rule
                name                            acme
                type                            uri-param
                action                          add
                match-val-type                  any
                match-value
                new-value                       "$LOCAL_IP"
```

For example, if the IP address of the SIP interface ($LOCAL_IP) is 10.1.2.3 and the Oracle Communications Session Border Controller receives the following Contact header:

```
Contact: <sip:1234@10.4.5.6;user=phone>
```

The header rule is applied to strip the user parameter from the Contact header URI and add the acme parameter with the value 10.1.2.3:

```
Contact: <sip:1234@10.4.5.6;acme=10.1.2.3>
```

Unique HMR Regex Patterns and Other Changes

In addition to the HMR support it offers, the SBC can now be provisioned with unique regex patterns for each logical remote entity. This supplement to pre-existing HMR functionality saves you provisioning time and saves SBC resources in instances when it was previously necessary to define a unique SIP manipulation per PBX for a small number of customer-specific rules.

The Default Expression

The SBC supports the non-standard regex `\,+` called the default expression. The default expression matches one or more characters, including NUL characters. The default expression cannot be used with other modifiers, like the star.

> **Note**
>
> In previous releases, the PCRE (Perl Compatible Regular Expression) engine used `\,` to match any character, including a NUL character. The PCRE engine was updated in 8.1 and no longer supports `\,`.

Manipulation Pattern Per Remote Entity

On the Oracle Communications Session Border Controller, you can configure logical remote entities (session agents, realms, and SIP interfaces) with a manipulation pattern string that the system uses as a regular expression. Then the SIP manipulation references this regular expression using the reserved word $MANIP_PATTERN. At runtime, the Oracle Communications Session Border Controller looks for the logical entity configured with a manipulation pattern string in this order of preference: session agent, realm, and finally SIP interface.

On finding the logical entity configured with the manipulation string, the Oracle Communications Session Border Controller dynamically determines the expression. When

there is an invalid reference to a manipulation pattern, the pattern-rule expression that results will turn out to be the default expression (which is \,+).

When the $MANIP_PATTERN is used in a manipulation rule's **new-value** parameter, it resolves to an empty string, equivalent of no value. Even though this process ends with no value, it still consumes system resources. And so Oraclerecommends you do not use $MANIP_PATTERN as a **new-value** value.

In the following example, the SIP manipulation references the regular expression from a realm configuration:

```
realm-config
        identifier                              net200
        description
        addr-prefix                             0.0.0.0
        network-interfaces                      public:0
        ...
        manipulation-pattern                    Lorem(.+)
sip-manipulation
        name                            manip
        description
        header-rules
                name                            headerRule
                header-name                     Subject
                action                          manipulate
                match-value                     $MANIP_PATTERN
                msg-type                        request
                comparison-type                 pattern-rule
                new-value                       Math
                methods                         INVITE
```

Reject Action

When you use this action type and a condition matching the manipulation rule arises, the Oracle Communications Session Border Controller rejects the request (though does not drop responses) and increments a counter.

- If the **msg-type** parameter is set to **any** and the message is a response, the Oracle Communications Session Border Controller increments a counter to show the intention to reject the message—but the message will continue to be processed.
- If the **msg-type** parameter is set to **any** and the message is a request, the Oracle Communications Session Border Controller performs the rejection and increments the counter.

The **new-value** parameter is designed to supply the status code and reason phrase corresponding to the reject. You can use the following syntax to supply this information: status-code[:reason-phrase]. You do not have to supply the status code and reason phrase information; by default, the system uses 400:Bad Request.

If you do supply this information, then the status code must be a positive integer between 300 and 699. The Oracle Communications Session Border Controller then provides the reason phrase corresponding to the status code. And if there is no reason phrase, the system uses the one for the applicable reason class.

You can also customize a reason phrase. To do so, you enter the status code followed by a colon (:), being sure to enclose the entire entry in quotation marks () if your reason code includes spaces.

When the Oracle Communications Session Border Controller performs the **reject** action, the current SIP manipulation stops processing and does not act on any of the rules following the **reject** rule. This course of action is true for nested SIP manipulations that might have been constructed using the **sip-manip** action type.

Reject Action Configuration

To support the **reject** action, set two parameters in the **session-router-config** configuration element.

1. Access the **session-router-config** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# session-router
ORACLE(session-router-config)#
```

2. **reject-message-threshold**—Enter the minimum number of message rejections allowed in the **reject-message-window** time on the SBC before generating an SNMP trap.

   The default is 0, meaning this feature is disabled and no trap will be sent.

3. **reject-message-window**—Enter the time in seconds that defines the window for maximum message rejections allowed before generating an SNMP trap.
4. Type **done** to save your configuration.

About Counters

The Oracle Communications Session Border Controller tracks messages that have been flagged for rejection using the **reject** action type. In the **show sipd** display, refer to the Rejected Messages category; there is no distinction between requests and responses.

```
ORACLE# show sipd
13:59:07-102
SIP Status                -- Period -- -------- Lifetime --------
                Active    High   Total      Total  PerMax    High
Sessions             0       0       0          0       0       0
Subscriptions        0       0       0          0       0       0
Dialogs              0       0       0          0       0       0
CallID Map           0       0       0          0       0       0
Rejections           -       -       0          0       0
ReINVITEs            -       -       0          0       0
Media Sessions       0       0       0          0       0       0
Media Pending        0       0       0          0       0       0
Client Trans         0       0       0          0       0       0
Server Trans         0       0       0          0       0       0
Resp Contexts        0       0       0          0       0       0
Saved Contexts       0       0       0          0       0       0
Sockets              0       0       0          0       0       0
Req Dropped          -       -       0          0       0
DNS Trans            0       0       0          0       0       0
DNS Sockets          0       0       0          0       0       0
DNS Results          0       0       0          0       0       0
Rejected Msgs        0       0       0          0       0       0
Session Rate = 0.0
```

```
Load Rate = 0.0
Remaining Connections = 20000 (max 20000)
```

SNMP Support

The Oracle Communications Session Border Controller provides SNMP support for the Rejected Messages data, so you can access this information externally. The new MIB objects are:

```
apSysRejectedMessages    OBJECT-TYPE
        SYNTAX          Counter32
        MAX-ACCESS      read-only
        STATUS          current
        DESCRIPTION
                "Number of messages rejected by the SD due to matching
criteria."
        ::= { apSysMgmtMIBGeneralObjects 18 }
apSysMgmtRejectedMesagesThresholdExeededTrap        NOTIFICATION-TYPE
        OBJECTS         { apSysRejectedMessages }
        STATUS          current
        DESCRIPTION
        " The trap will be generated when the number of rejected messages
exceed the configured threshold within the configured window."
        ::= { apSystemManagementMonitors 57 }
apSysMgmtRejectedMessagesGroup  OBJECT-GROUP
        OBJECTS {
            apSysRejectedMessages
        }
        STATUS          current
        DESCRIPTION
                "Objects to track the number of messages rejected by the SD."
        ::= {  apSystemManagementGroups 18 }
apSysMgmtRejectedMessagesNotificationsGroup NOTIFICATION-GROUP
      NOTIFICATIONS {
                        apSysMgmtRejectedMesagesThresholdExeededTrap
                      }
                STATUS          current
                DESCRIPTION
                "Traps used for notification of rejected messages"
      ::= {  apSystemManagementNotificationsGroups 26 }
apSmgmtRejectedMessagesCap
            AGENT-CAPABILITIES
            PRODUCT-RELEASE     "Acme Packet SD"
            STATUS               current
            DESCRIPTION         "Acme Packet Agent Capability for enterprise
                                 system management MIB."
            SUPPORTS            APSYSMGMT-MIB
                INCLUDES        {
                                  apSysMgmtRejectedMessagesGroup,
                                  apSysMgmtRejectedMessagesNotificationsGroup
                                }
            ::= { apSmgmtMibCapabilities 37 }
```

Log Action

When you use this action type and a condition matching the manipulation rule arises, the SBC logs information about the current message to a separate log file. This log files will be located on the same core in which the SIP manipulation occurred. On the core where sipt runs, a logfile called matched.log will appear when this action type is executed.

The matched.log file contains a timestamp, received and sent SBC network interface, sent or received IP address:port information, and the peer IP address:port information. It also specifies the rule that triggered the log action in this syntax: rule-type[rule:name]. The request URI, Contact header, To Header, and From header are also present.

```
---------------------------------------
Apr 17 14:17:54.526 On [0:0]192.168.1.84:5060 sent to 192.168.1.60:5060
element-rule[checkRURIPort]
INVITE sip:service@192.168.1.84:5060 SIP/2.0
From: sipp <sip:+2125551212@192.168.1.60:5060>;tag=3035SIPpTag001
To: sut <sip:service@192.168.1.84>
Contact: sip:sipp@192.168.1.60:5060
```

Changes to Storing Pattern Rule Values

Release S-C6.2.0 introduces changes to the framework for storing regular expression results within manipulation rules, altering the way the **store** action works. These changes are beneficial to performance.

In previous releases, when the **store** action is used, the Oracle Communications Session Border Controller stores all values matching the regular expression defined in the **match-value** parameter for all headers. At runtime, the system evaluates all stored values to find the correct index.

Now, you no longer need to specify the **store** action. The simple fact of referencing another rule tells the system it must store a value. When SIP manipulation is used, the system first checks to see if any values require storing. The **add** action is an exception to this process; storing happens after a header is added.

When referring to a rule, that rule still needs to have a regular expression defined in the match-vale and the comparison type set to pattern-rule; else the default expression will be used.

Removal of Restrictions

The following restrictions related to HMR have been removed in Release S-C6.2.0:

- The action **find-replace-all** now executes all element rules. Previously, no child rules were executed.
- The action **sip-manip** now executes existing all element rules. Previously, no child rules were executed.
- The action **store** now executes existing all element rules. Previously, only child rules with the **store** action were executed.
- The action **add** now executes existing all element rules. Previously, only child rules with the **add** action were executed.

Name Restrictions for Manipulation Rules

Historically, you have been allowed to configure any value for the name parameter within a manipulation rule. This method of naming caused confusion when referencing rules, so now manipulation rules name must follow a specific syntax. They must match the expression ^[[alpha:]][[:alnum:]_]+$ and contain at least one lower case letter.

In other words, the name must:

- Start with a letter, and then it can contain any number of letters, numbers, or underscores
- Contain at least one lower case letter

All pre-existing configurations will continue to function normally. If you want to change a manipulation rule, however, you are required to change its name if it does not follow the new format.

The ACLI **verify-config** command warns you if the system has loaded a configuration containing illegal naming syntax.

Please note that the software allows you to make changes to HMRs, including configuring new functionality to existing rules, as long as you do not change the rule name. This results in an important consideration surrounding HMRs with hyphens in previously configured rule names.

- You can reference stored values in new value names. (Recall that stored values may be rule names.)
- You can perform subtraction in new value names.

If you use a rule names with hyphens within the REGEX of new value names, the system cannot determine whether the hyphen is part of the rule name or is intended to invoke subtraction within the REGEX. For this reason, you need to use great care with legacy HMR naming that includes hyphens.

As a general rule, create new rule names that follow the new rule naming guidelines if you intend to use new functionality in those rules.

New Value Restrictions

To simplify configuration and remove possible ambiguity, the use of boolean and equality operators (==, <=, <, etc.) for **new-value** parameter values has been banned. Since there was no specific functionality tied to their use, their ceasing to be use will have no impact to normal SIP manipulation operations.

MIME Support

You can manipulate MIME types in SIP message bodies. You can manipulate the body of SIP messages or a specific content type and you can change the MIME attachment of a specific type within the body by using regular expressions. You search for a particular string and the replacement of all matches for that type using a find-replace-all action.

> **Note**
>
> The find-replace-all action can consume more system resources than other HMR types of action. Use this powerful action type only when another action cannot perform the type of manipulation you require.

Manipulating MIME Attachments

Set the action type to find-replace-all to modify MIME attachments.

To manipulate a particular portion of the MIME attachment, for example when removing a certain attribute within the Content-Type of `application/sdp`, the SBC needs to search the content multiple times because:

- SDP can have more than one media line
- The SIP message body can contain more than one `application/sdp`.

When the action type is find-replace-all, the SBC treats the match-value as a regular expression and binds the comparison-type to pattern-rule, even if comparison-type is set to some other value. This type of action is both a comparison and action: for each regular expression match within the supplied string, the SBC substitutes the new value for that match.

Use subgroups to replace portions of the regular expression rather than the entire matched expression. The subgroup replacement syntax is formed by adding the string `[[:n:]]` to the end of the regular expression—where `n` is a number between 0 and 9. For example, setting the following parameters

```
action                    find-replace-all
match-value               sip:(user)@host[[:1:]]
new-value                 bob
```

creates a new rule to replace only the user portion of the URI that searches for the regular expression and replaces all instances of the user subgroup with the value bob.

Setting the following parameters

```
action                    find-replace-all
match-value               0
new-value                 1
```

creates a new rule to recursively replace all the 0 digits in a telephone number with 1. With this rule the user portion of a URI—or for any other string—with a value 1-781-308-4400 would be replaced as 1-781-318-4411.

If you leave the **new-value** parameter blank for **find-replace-all**, the SBC replaces the matched sub-group with an empty string—an equivalent of deleting the sub-group match. You can also replace empty sub-groups, which is like inserting a value within the second sub-group match. For example, `user()@example.com[[:1:]]` with a configured new-value `_bob` yields `user_bob@host.com`.

Setting **find-replace-all** disables the following **parameter-type** values: **uri-param-name**, **uri-header-name**, and **header-param-name**. These values are unusable because the SBC only uses case-sensitive matches for the match-value to find the parameter name within the URI. Since it can only be found by exact match, the SBC does not support finding and replacing that parameter.

About the MIME Value Type

To modify the MIME attachment, the SBC supports a `mime` value for the type parameter in the element rules. You can only use the `mime` type value against a specific header, which in this case is Content (abbreviated as c).

When you set the element rule type to `mime`, you must also set a value for the parameter-name. This step is a requirement because it sets the content-type the SBC manipulates in a specific part of the MIME attachment. You cannot leave this parameter blank; the SBC does not let you save the configuration if you do. When you use the store action on a multi-part MIME attachment that has different attachment types, the SBC stores the final instance of the content-type because it does not support storing multiple instances of element rule stored values.

If you do not know the specific content type, which means the SBC will find the match value, you can use the asterisk * as a wildcard with the parameter-name. (You cannot, however, set partial content types, for example, `application/*`.) The SBC then loops through the MIME attachment's content types.

MIME manipulation does not support manipulating headers in the individual MIME attachments. For example, the SBC cannot modify the Content-Type given a portion of a message body like this one:

```
--boundary-1
Content-Type: application/sdp
v=0
o=use1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=audio 10000 RTP/AVP 8
a=rtpmap:8 PCMA/8000/1
a=sendrecv
a=ptime:20
a=maxptime:200
```

SIP Message-Body Separator Normalization

The stripPreambleCrlf option normalizes CLRF message-body separators.

The SBC supports MIME attachments — up to a maximum payload size of 64KB — and has the ability to allow more than the required two CRLFs between the SIP message headers and the multipart body's first boundary. The first two CRLFs that appear in all SIP messages signify the end of the SIP header and the separation of the header and body of the message, respectively. Sometimes additional extraneous CRLFs can appear within the preamble before any text.

The SBC works by forwarding received SIP messages regardless of whether they contain two or more CRLFs. Although three or more CRLFs are legal, some SIP devices do not accept more than two.

To ensure all SIP devices accept messages from the SBC, strip all CRLFs located at the beginning of the preamble before the appearance of any text, ensuring that there are no more than two CRLFs between the end of the last header and the beginning of the body within a SIP message. Enable this feature by adding the new stripPreambleCrlf option to the global SIP configuration.

To enable the stripping of CRLFs in the preamble:

1. Access the **sip-config** configuration element.

   ```
   ORACLE# configure terminal
   ORACLE(configure)# session-router
   ```

```
ORACLE(session-router)# sip-config
ORACLE(sip-config)#
```

2. **options**—Set the options parameter by typing options, a Space, the option name stripPreambleCrlf with a plus sign.

   ```
   ORACLE(sip-config)# options +stripPreambleCrlf
   ```

   In order to append the new options to the global SIP configuration's options list, you must prepend the new option with a plus sign. If you type the option without the plus sign, you will overwrite any previously configured options.

3. Save and activate your configuration.

Configuring MIME Support

To enable MIME support, set the **action** configuration element to **find-replace-all** at both the header-rules level and element-rules level. Set the **type** configuration element to **mime** at the element-rules level.

1. Access the **header-rules** configuration element.

   ```
   ORACLE# configure terminal
   ORACLE(configure)# session-router
   ORACLE(session-router)# sip-manipulation
   ORACLE(sip-manipulation)# header-rules
   ORACLE(sip-header-rules)#
   ```

2. **action**—Enter **find-replace-all**.

   ```
   ORACLE(sip-header-rules)# action find-replace-all
   ```

3. Navigate to the **element-rules** configuration element.

   ```
   ORACLE(sip-header-rules)# element-rules
   ORACLE(sip-element-rules)#
   ```

4. **action**—Enter **find-replace-all**.

   ```
   ORACLE(sip-element-rules)# action find-replace-all
   ```

5. **type**—Enter **mime**.

   ```
   ORACLE(sip-element-rules)# type mime
   ```

6. Save and activate your configuration.

HMR for SIP-ISUP

You can apply HMRs on ISDN user party (ISUP) binary bodies. Using the same logic and mechanisms applied to SIP header elements, HMR for SIP-ISUP manipulates ISUP parameter fields and ISUP message parts. You create MIME rules that manipulate targeted body parts of a SIP message.

MIME Rules Overview

MIME rules operate much the same way that SIP header rules do. You can set parameters in the MIME rules that the SBC uses to match against specific SIP methods and message types. The system compares the search criteria against the body or body parts using the type of comparison you choose. You can pick the kind of manipulation that suits your needs; the SBC then takes action with matching and new values to change the SIP message.

> **Note**
>
> Using the delete action on a multi-part MIME string reduces a number of bodies down to one and the SIP message remains a multi-part MIME message with only one body part (and thereby avoids the header conflicting with the message itself).

Identifying a MIME Rule

You identify the MIME rule by using a content type that refers to the specific body part on which to operate. For example, given a SIP Content-Type header with the value `multipart/mixed;boundary=unique-boundary-1`, you would enter a content type value of `application/sdp` to specifically manipulate the SDP portion of the SIP message. The SBC knows automatically if it is operating on SIP messages with single or multiple body parts, and the content type applies to both kinds. When making its comparison, the SBC matches the content type of the body without regard to case (case insensitive), ignoring any header parameters.

Both for making comparisons against the body part and for new/replacement values, the SBC treats the match and new values you set for a MIME rule as ASCII strings. A MIME rule operating on a binary body part yields an improper conversion of a new value with respect to the binary body part.

About MIME Rules

MIME rules (set up in the ACLI **mime-rules** configuration) operate much the same way that SIP header rules do. You can set parameters in the MIME rules that the Oracle Communications Session Border Controller uses to match against specific SIP methods and message types. The system compares the search criteria against the body or body parts using the type of comparison you choose. Offering a variety of selection, you can pick kind of manipulation that suits your needs; the Oracle Communications Session Border Controller then takes action with matching and new values to change the SIP message.

> **Note**
>
> when you use the **delete** action on a multi-part MIME string that reduces a number of bodies down to one, the SIP message remains a multi-part MIME message with only one body part (and thereby avoids the header conflicting with the message itself).

You identify the MIMe rule by configuring a content type that refers to the specific body part on which to operate. For example, given a SIP Content-Type header with the value multipart/mixed;boundary=unique-boundary-1, you would enter a content-type value of **application/sdp** to manipulate specifically on the SDP portion of the SIP message. The Oracle Communications Session Border Controller knows automatically if it is operating on SIP

messages with single or multiple body parts, and the content-type setting applies to both kinds. And when making its comparison, the Oracle Communications Session Border Controller matches the content-type of the body with regard to case (case insensitive), ignoring any header parameters.

Both for making comparisons against the body part and for new/replacement values, the Oracle Communications Session Border Controller treats the match and new values you set for a MIME rule as ASCII strings. Therefor, a mime rule operating on a binary body part will yield an improper conversion of a new value with respect to the binary body part.

Within MIME rules, you configure MIME headers, which operate on the specific headers in the match body part of the SIP message. The Oracle Communications Session Border Controller uses the MIME header name to run a string comparison to match the specific header in the message's body part.

Using these rules, you can also manipulate the preamble—or the SIP message text that follows the headers but precedes the body separator. To do so, enter the keyword **@preamble** for the content type parameter in the MIME rule. Likewise you can manipulate the epilogue—or the text that follows the last body part after the last separator—using the keyword **@epilogue**.

Note that the ACLI limits character entries to 255 characters before the return character must be entered, but MIME parts can easily exceed this 255-character size. So you might need to enter a value larger that 255 characters. To do so, you start your entry (in the match-value or new-value parameters) with a plus sign (+). The plus sign instructs the system to add the string after it to the pre-existing match or new value. For the new-value parameter, the Oracle Communications Session Border Controller checks the value immediately for validity. Be sure that when you are appending values to a new-value that the entire expression is valid at each point where strings are appended.

MIME Rules Configuration

This section shows you how to configure MIME rules and MIME headers.

To configure MIME rules:

1. Access the **sip-manipulation** configuration element.

```
ORACLE# configure terminal
ORACLE(configure)# session-router
ORACLE(session-router)# sip-manipulation
ORACLE(sip-manipulation)#
```

2. If you are adding this feature to an existing configuration, remember you must select the configuration you want to edit.

```
ORACLE(sip-manipulation)# select
<name>:
1: name= addRemoteIP  desc=
2: name= addTargetIP  desc=

selection: 2
ORACLE(sip-manipulation)#
```

3. Access the **mime-rules** configuration element.

```
ORACLE(sip-manipulation)# mime-rules
ORACLE(sip-mime-rules)#
```

4. **name**—Enter a name for this MIME rule. This parameter is required and has no default.
5. **content-type**—Enter the content type for this MIME rule. This value refers to the specific body part in the SIP message body that is to be manipulated. For example, given a SIP Content-Type header with the value multipart/mixed;boundary=unique-boundary-1, you would enter a content-type value of **application/sdp** to manipulate specifically on the SDP portion of the SIP message.

   To manipulate the SIP preamble or epilogue, enter the keyword **@preamble** or keyword **@epilogue**.
6. **action**—Choose the type of action you want to be performed: **none**, **add**, **delete**, **manipulate**, **store**, **sip-manip**, **find-replace-all**, **reject**, **log** and **monitor**. These are the same actions you can select when configuring SIP header manipulation. The default is **none**.
7. **comparison-type**—Enter the way that you want body part of the SIP message to be compared. This choice dictates how the Oracle Communications Session Border Controller processes the match rules against the SIP header. the default is **case-sensitive**. The valid values are: **case-sensitive**, **case-insensitive**, **boolean**, **refer-case-sensitive**, **refer-case-insensitive**, and **pattern-rule**.
8. **msg-type**—Enter the SIP message type on which you want the MIME rules to be performed. Valid values are **any**, **request**, **reply** and **out-of-dialog**. The default value is **any**.
9. **methods**—Enter the list of SIP methods to which the MIME rules applies. There is no default for this parameter.
10. **match-value**—Enter the value to match against the body part in the SIP message. This is where you can enter values to match using regular expression values. Your entries can contain Boolean operators.
11. **new-value**—When the action parameter is set to **add** or to **manipulate**, enter the new value that you want to substitute.

    To configure MIME headers for performing HMR operations on specific headers in the matched body part of the SIP message:
12. Follows Steps 1 through 4 above.
13. **Type mime-header-rules and press Enter.**

    ```
    ORACLE(sip-mime-rules)# mime-header-rules
    ORACLE(sip-mime-header-rules)#
    ```
14. **name**—Enter a name for this MIME header rule. This parameter is required and has no default.
15. **mime-header**—Enter the value to be used for comparison with the specific header in the body part of the SIP message. There is no default for this parameter.
16. **action**—Choose the type of action you want to be performed: **none**, **add**, **store**, **sip-manip**, **replace**, **find-replace-all**, **delete**,**log**,**monitor** and **reject**. The default is **none**.
17. **comparison-type**—Enter the way that you want the header in the body part of the SIP message to be compared. This choice dictates how the Oracle Communications Session Border Controller processes the match rules against the SIP header. the default is **case-sensitive**. The valid values are: **case-sensitive**, **case-insensitive**, **boolean**, **refer-case-sensitive**, **refer-case-insensitive**, and **pattern-rule**.

18. **match-value**—Enter the value to match against the header in the body part of the SIP message. This is where you can enter values to match using regular expression values. Your entries can contain Boolean operators.
19. **new-value**—When the action parameter is set to **add** or to **manipulate**, enter the new value that you want to substitute.
20. Save your work.

Working with MIME Rules

Within MIME rules, you configure MIME headers that operate on the specific headers in the match body part of the SIP message. The SBC uses the MIME header name to run a string comparison to match the specific header in the message's body part.

Using these rules, you can also manipulate the preamble or the SIP message text that follows the headers but precedes the body separator. To do so, enter the keyword `@preamble` for the content type parameter in the MIME rule. Likewise you can manipulate the epilogue or the text that follows the last body part after the last separator using the keyword `@epilogue`.

The ACLI limits character entries to 255 characters before the return character must be entered. MIME parts can easily exceed this 255-character size, so you might need to enter a value larger that 255 characters. To do so, you start your entry with a plus sign +. The plus sign instructs the system to add the string after it to the pre-existing match or new value. For the new-value parameter, the SBC checks the value immediately for validity. Be sure that when you are appending values to a new-value that the entire expression is valid at each point where strings are appended.

MIME ISUP Manipulation

ISUP message can be carried in SIP messages through either a standard body or through a multipart MIME encoded body. While ANSI and ITU are the two major groups, each contains many specific variants. To facilitate instances where two sides of a call use different versions, the SBC supports interworking between the following SIP ISUP formats: ANSI, ITU, ETSI-356 (an ITU variant), and GR-317 (an ANSI variant). To do so, the SBC can move, delete, and add parameters to various sections of the message.

The ISUP message version is determined by either the content type of the SIP message or the MIME content-type. Messages that contain an unknown ISUP format pass through the SBC untouched. You can perform HMR operations on SIP ISUP binary bodies (MIME ISUP).

> **Note**
>
> Custom formats are not supported.

Within **mime-isup-rule**, **isup-param-rule**, the **format** field instructs the SBC how to encode and decode the current string. The field options are hexascii, binary-ascii, ascii-string, bcd, and number-param.

- `hex-ascii`—the SBC will decode the ISUP param string from its binary value in the SIP message into a string of hexadecimal ASCII (as seen in Wireshark) before applying the match-value. It will convert the resolved new-value from hex-ascii into binary into the message. For example, if the received ISUP param was the binary of `0x010a`, it will convert it into the string `010a`, and then apply the match-value. If the regex pattern is `^01` then it would match, as would `0a$` and `^010a$`. If the new-value is `010b`, then it will encode it into

the binary `0x010b`. Since this is done after resolving the new-value. The new-value can reference a previously stored value as long as it is hex-ascii format.

- `binary-ascii`— the SBC will decode the ISUP param string from its binary value in the SIP message into a string of ones and zeros representing the individual bits. It will convert the new-value as long as it's ones and zeros within the param. For example, if the received ISUP param was the binary `0x010a`, it will convert it into the string `0000000100001010`, and then apply the match-value. If the regex pattern is `^.......(.)` or `^.{7}(.)` then in both cases it will store the 8th bit value in `$1`. In this manner, the user can check, get, or set individual bits in parameters. The new-value can be a string, reference a stored value, or be a concatenation of them as long as it is ones and zeros after being resolved.
- `ascii-string`—the SBC will decode the ISUP param string from its binary value in the SIP message into an ASCII string based on the ASCII specification and convert the new-value back. For example, if the received ISUP param was the binary `0x4849`, it will convert it into the string `HI`, and then apply the match-value.
- `bcd`—the SBC will decode the ISUP param string from its binary value in the SIP message into digits using the BCD variant of ISUP. For example, if the received ISUP param was the binary `0x0123`, it will convert it to the string `0123` and then apply the match-value.
- `number-param`—the SBC will decode the ISUP param string from its binary value in the SIP message into a string representation of an E.164 phone number. The ISUP param must be in a number formatted parameter like Calling Party Number or Called Party Number. The SBC treats the ISUP parameter as one of the common number parameter formats: the SBC will automatically decode the correct number of digits based on the odd/even bit in the parameter, and add a leading + based on the Nature of Address (NoA) field being E.164 international. Similarly, when the SBC converts the new-value back into the ISUP parameter, it will set the odd/even bit correctly, and set the NoA field based on the existence of the leading + character. The string applied to match-value thus looks the same as an element-rule of type phone-number (i.e. +12125551212). Since this format is specific to ISUP parameters, it can only be used in isup-param-rule.

Adding an ISUP Body to a SIP Message

Unlike the MIME manipulation you can use by setting the SIP header rules accordingly, you can add MIME parts to SIP messages using the MIME rules configuration.

You can configure a SIP header manipulation to add an ISUP body to a SIP message. and the Oracle Communications Session Border Controller adds them after any SDP parts if they are present. You can add an ISUP body to a SIP message in two ways:

- You can create a **mime-isup-rule** with the **action** type set to **add**, and enter the entire body in string hexadecimal form in the **new-value** parameter.
- You can leave the **new-value** parameter empty at the **mime-isup-rule** level and create an add rule for an **isup-param-rule**.
  In this case, the Oracle Communications Session Border Controller creates the corresponding ISUP message based on the **isup-msg-type** value and supply all of the parameters with their default values. Since the **isup-msg-type** takes a list of values as a valid entry, for this case it only uses the first one. However, the Oracle Communications Session Border Controller ignores the **isup-msg-type** value if you set the **new-value** parameter. And the **isup-param-rule**, if configured, overwrite the default value or add a new parameter based on the defined parameter type.

  It is also possible that you might supply a **new-value** both at the **mime-isup-rule** level and at the **isup-param-rule** level. If you do, the **new-value** entry from the **mime-isup-rule** is parsed into an ISUP object and the **isup-param-rule** operates on that object.

MIME ISUP Manipulation Configuration

This section shows you how to configure MIME ISUP manipulation.

1. In Superuser mode, type **configure terminal** and press Enter.

   ```
   ORACLE# configure terminal
   ORACLE(configure)#
   ```

2. Type **session-router** and press Enter.

   ```
   ORACLE(configure)# session-router
   ORACLE(session-router)#
   ```

3. Type **sip-manipulation** and press Enter. **If you are adding this feature to an existing configuration, then remember you must select the configuration you want to edit.**

   ```
   ORACLE(session-router)# sip-manipulation
   ORACLE(sip-manipulation)#
   ```

4. **Type mime-isup-rules and press Enter.**

   ```
   ORACLE(sip-manipulation)# mime-isup-rules
   ORACLE(sip-mime-isup-rules)#
   ```

5. **name**—Enter a name for this MIME ISUP rule. This parameter is required and has no default.
6. **content-type**—Enter the content type for this MIME rule. This value refers to the specific body part in the SIP message body that is to be manipulated. For example, given a SIP Content-Type header with the value multipart/mixed;boundary=unique-boundary-1, you would enter a content-type value of **application/sdp** to manipulate specifically on the SDP portion of the SIP message.

   To manipulate the SIP preamble or epilogue, enter the keyword **@preamble** or keyword **@epilogue**.
7. **action**—Choose the type of action you want to be performed: **none**, **add**, **delete**, **manipulate**, **store**, **sip-manip**, **find-replace-all**, **reject**,**log** and **monitor**. These are the same actions you can select when configuring SIP header manipulation. The default is **none**.
8. **comparison-type**—Enter the way that you want body part of the SIP message to be compared. This choice dictates how the Oracle Communications Session Border Controller processes the match rules against the SIP header. the default is **case-sensitive**. The valid values are: **case-sensitive**, **case-insensitive**, **boolean**, **refer-case-sensitive**, **refer-case-insensitive**, and **pattern-rule**.
9. **msg-type**—Enter the SIP message type on which you want the MIME rules to be performed. Valid values are **any**, **request**, **reply** and **out-of-dialog**. The default value is **any**.
10. **methods**—Enter the list of SIP methods to which the MIME rules applies. There is no default for this parameter.
11. **match-value**—Enter the value to match against the body part in the SIP message. This is where you can enter values to match using regular expression values. Your entries can contain Boolean operators.

12. **new-value**—When the action parameter is set to **add** or to **manipulate**, enter the new value that you want to substitute.
13. **isup-spec**—Specify how the Oracle Communications Session Border Controller is to parse the binary body; valid values are the enumerated type. The values for this parameter are these SIP ISUP formats:
    - **ANSI-2000** (default)—Corresponding to ANSI T1.113-2000
    - **ITU-99**—Corresponding to ITU Q.763
14. **isup-msg-type**—Identify the specific ISUP message types (such as IAM and ACM) on which to operate. The Oracle Communications Session Border Controller uses with the **msg-type** parameter (which identifies the SIP message) in the matching process. You enter values in this parameters as a list of numbers rather than as an enumerated value because of the large number of ISUP message type, and the range is between 0 and 255. There is no default for this parameter.
15. **mime-header**—Enter the value to be used for comparison with the specific header in the body part of the SIP message. There is no default for this parameter.

    To configure ISUP parameters rules:
16. Follows Steps 1 through 4 above.
17. **Type isup-parameter-rules and press Enter.**

    ```
    ORACLE(sip-mime-isup-rules)# isup-param-rules
    ORACLE(sip-isup-param-rules)#
    ```
18. **name**—Enter a name for this ISUP parameter rule. This parameter is required and has no default.
19. **mime-header**—Enter the value to be used for comparison with the specific header in the body part of the SIP message. There is no default for this parameter.
20. **action**—Choose the type of action you want to be performed: **none**, **add**, **delete**, **store**, **sip-manip**, **replace**, **find-replace-all**, **log**, **monitor** and **reject**. The default is **none**.
21. **comparison-type**—Enter the way that you want the header in the body part of the SIP message to be compared. This choice dictates how the Oracle Communications Session Border Controller processes the match rules against the SIP header. the default is **case-sensitive**. The valid values are: **case-sensitive**, **case-insensitive**, **boolean**, **refer-case-sensitive**, **refer-case-insensitive**, and **pattern-rule**.
22. **match-value**—Enter the value to match against the header in the body part of the SIP message. This is where you can enter values to match using regular expression values. Your entries can contain Boolean operators.
23. **new-value**—When the action parameter is set to **add** or to **manipulate**, enter the new value that you want to substitute.
24. **parameter-type**—Using ISUP parameter mapping, enter which of the ISUP parameters on which your want to perform manipulation. This parameter takes values between 0 and 255, and you must know the correct ISUP mapping value for your entry. The Oracle Communications Session Border Controller calculates the offset and location of this parameter in the body. Note that the value returned from the body does not the type or length, only the parameter value. For example, a parameter-type value of 4 acts on the Called Party Number parameter value.
25. parameter-format—Enter how you want to convert specific parameter to a string representation of that value. Valid values for **parameter-format** are: **number-param**, **hex-ascii** (default), **binary-ascii**, **ascii-string**, and **bcd**. Both match and new values are

encoded and decoded by the designated parameter-format type. In this regard, the **match-value** decodes the parameters and the **new-value** encodes the ASCII string into the respective binary format.

26. Save your work.

Configuration Example

This section provides an example of a SIP manipulation configuration that shows MIME rules and MIME ISUP rules.

```
sip-manipulation
        name                            manip
        description
        header-rule
                name                            headerRule1
                header-name                     Date
                action                          add
                comparison-type                 case-sensitive
                msg-type                        reply
                methods
                match-value
                new-value
                element-rule
                        name                            elemRule1
                        parameter-name
                        type                            header-value
                        action                          add
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-value                       "August 19, 1967"
mime-rule
                name                            mimeRule1
                Content-Type                    application/SDP
                action                          manipulate
                comparison-type                 case-sensitive
                msg-type                        request
                methods
                match-value
                new-value
                mime-header
                        name                            mimeHeaderRule1
                        mime-header-name                Content-Disposition
                        action                          add
comparison-type                 case-sensitive
                        match-value
                        new-value                       "signal;
handling=required"
        mime-isup-rule
                name                            mimeRule1
                content-type                    application/ISUP
                action                          manipulate
                comparison-type                 case-sensitive
                msg-type                        request
                methods                         INVITE
```

```
                match-value
                new-value
                isup-spec                       {ansi00, itu-92}
                isup-msg-type                   0 (0-256 IAM, ACM, etc.)
                mime-header
                        name                            mimeHeaderRule1
                        mime-header-name                Content-Disposition
                        action                          add
                        comparison-type                 case-sensitive
                        match-value
                        new-value                       "signal;
handling=optional"
                isup-param-rule
                        name                            isupRule1
                        parameter-type                  # {0-256 specific type)
                        parameter-format                {number-parameter,
hex, binary, ascii, bcd}
                        action                          add
                        comparison-type                 case-sensitive
                        match-value
                        new-value                       "signal;
handling=optional"
```

Header Manipulation Rules for SDP

The SBC supports SIP header and parameter manipulation rules for four types of SIP message contents:

- headers
- elements within headers
- ASCII-encoded Multipurpose Internet Mail Extensions (MIME) bodies
- binary-encoded MIME ISDN User Part (ISUP) bodies

While Session Description Protocol (SDP) offers and answers can be manipulated in a fashion similar to ASCII-encoded MIME, such manipulation is primitive in that it lacks the ability to operate at the SDP session- and media-levels.

In addition, the system supports a variant of Header Manipulation Rules (HMR) operating on ASCII-encoded SDP bodies, with specific element types for descriptors at both the session-level and media-level, and the ability to apply similar logic to SDP message parts as is done for SIP header elements.

The configuration object, mime-sdp-rules, under sip-manipulation specifically addresses the manipulation of SDP parts in SIP messages. Just as existing header-rules are used to manipulate specific headers of a SIP message, mime-sdp-rules will be used to manipulate the SDP specific mime-attachment of a SIP message.

SDP Manipulation

mime-sdp-rules function in a similar fashion as header-rules. They provide

- parameters used to match against specific SIP methods and/or message types
- parameters used to match and manipulate all or specified parts of an SDP offer or answer
- a means of comparing search strings or expressions against the entire SDP

- different action types to allow varying forms of manipulation

Since only a single SDP can exist within a SIP message, users need not specify a content-type parameter as is necessary for a mime-rule. A mime-sdp-rule operates on the single SDP within the SIP message. If no SDP exists with the message, one can be added. If the message already contains a mime attachment, adding SDP results in a multipart message.

All manipulations performed against all or parts of the SDP are treated as UTF-8 ASCII encoded text. At the parent-level (mime-sdp-rule) the **match-value** and **new-value** parameters execute against the entire SDP as a single string.

An add action only succeeds in the absence of SDP because a message is allowed only a single SDP offer or answer. A delete operation at the mime-sdp-rule level will remove the SDP entirely.

Note that on an inbound sip-manipulation, SDP manipulations interact with the SBC codec-policy. SDP manipulations also interact with codec reordering and media setup. It is very possible to make changes to the SDP such that the call can not be setup due to invalid media parameters, or settings that will affect the ability to transcode the call. Consequently, user manipulation of the SDP can prove risky, and should be approached with appropriate caution.

Three configuration-objects, sdp-session-rule, sdp-media-rule, and mime-header-rule, exist under the mime-sdp-rule. These objects provide finer grained control of manipulating parts of the SDP.

sdp-session-rule

An sdp-session-rule groups all SDP descriptors, up until the first media line, into a single entity, thus allowing the user to perform manipulation operations on a session-specific portion of the SDP.

Like the mime-sdp-rule, all match-value and new-value operations performed at this level are executed against the entire session group as a complete string. Given the sample SDP below, if an sdp-session-rule is configured, the match-value and new-values operate only on the designated portion.

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
a=orient:portrait
```

Nested under the sdp-session-rule configuration object is an sdp-line-rule object, the object that identifies individual descriptors within the SDP. The types of descriptors used at the sdp-session-rule level are v, o, s, i, u, e, p, c, b, t, r, z, k, and a, the descriptors specific to the entire session description.

This level of granularity affords the user a very simple way to making subtle changes to the session portion of the SDP. For instance, it is very common to have to change the connection line at the session level.

The add and delete actions perform no operation at the sdp-session-rule level.

sdp-media-rule

An sdp-media-rule groups all of the descriptors that are associated with a specific media-type into single entity, thus allowing the user to perform manipulation operations on a media-specific portion of the SDP. For example, a user can construct an sdp-media-rule to change an attribute of the audio media type.

Like a mime-sdp-rule, all match-value and new-value operations performed at this level are executed against the entire media-group as a complete string. Given the sample SDP below, if a media-level-descriptor is configured to operate against the application group, the match-value and new-values would operate only on designated portion.

```
        v=0
        o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
        s=SDP Seminar
        i=A Seminar on the session description protocol
        u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
        e=mjh@isi.edu (Mark Handley)
        c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
        a=recvonly
        m=audio 49170 RTP/AVP 0
        m=video 51372 RTP/AVP 31
        m=application 32416 udp wb
        a=orient:portrait
```

A configuration parameter **media-type** is used to specify the media group on which to operate. It contains all of the descriptors including the m-line up to the next m-line. This parameter is a string field and must match the media-type exactly as it appears within the SDP. The special media-type media can be used to refer to all media types. This is particularly useful when performing an add operation, when the user wants to add a media section between the first and second medias, but does not know what media type they are. Otherwise, during an add operation, the media section would be added before the specified media-type (if no index parameter was provided).

The types of descriptors used at the sdp-media-rule level are m, i, c, b, k, and a, the descriptors specific to the media description.

This level of granularity affords the user a very simple way to making subtle changes to the media portion of the SDP. For instance, it is very common to have to change the name of an audio format (for example G729 converted to g729b), or to add attributes specific to a certain media-type.

The index operator is supported for the media-type parameter (for example, media-type audio[1]). Like header rules, if no index is supplied, this means operate on all media-types that match the given name. For specifying specific media-types, the non-discrete indices are also supported (for example, ^ - last). Adding a media-type, without any index supplied indicates that the media should be added at the beginning. The special media-type media uses the index as an absolute index to all media sections, while a specific media-type will index relative to that given media type.

For sdp-media-rules set to an action of add where the media-type is set to media, the actual media type is obtained from the new-value, or more specifically, the string after m= and before the first space.

Given the following SDP:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
m=audio 48324 RTP/AVP 8
m=video 51372 RTP/AVP 31
```

With the sdp-media-rule:

```
sdp-media-rule
        name                        smr
        media-type                  audio[1]
        action                      manipulate
        comparison-type             case-sensitive
        match-value
        new-value                   "m=audio 1234 RTP/AVP 8 16"
```

This rule operates on the 2nd audio line, changing the port and adding another codec, resulting in the SDP:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
m=audio 1234 RTP/AVP 8 16
m=video 51372 RTP/AVP 31
```

The following rule, however:

```
sdp-media-rule
        name                        smr
        media-type                  media[1]
        action                      add
        comparison-type             case-sensitive
        match-value
        new-value                   "m=video 1234 RTP/AVP 45"
```

adds a new video media-type at the 2nd position of all media-lines, resulting in the SDP:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
m=video 1234 RTP/AVP 45
m=audio 48324 RTP/AVP 8
m=video 51372 RTP/AVP 31
```

sdp-line-rule

Unlike header-rules, sdp descriptors are not added in the order in which they are configured. Instead they are added to the SDP adhering to the grammar defined by RFC 4566 (as is shown below).

```
Session description
   v=  (protocol version)
   o=  (originator and session identifier)
   s=  (session name)
   i=* (session information)
   u=* (URI of description)
   e=* (email address)
   p=* (phone number)
      c=* (connection information -- not required if included in
        all media)
   b=* (zero or more bandwidth information lines)
   One or more time descriptions ("t=" and "r=" lines; see
        below)
   z=* (time zone adjustments)
   k=* (encryption key)
   a=* (zero or more session attribute lines)
   Zero or more media descriptions (see below)

Time description
   t=  (time the session is active)
   r=* (zero or more repeat times)

Media description, if present
   m=  (media name and transport address)
   i=* (media title)
   c=* (connection information -- optional if included at
        session level)
   b=* (zero or more bandwidth information lines)
   k=* (encryption key)
   a=* (zero or more media attribute lines)
```

* after the equal sign denotes an optional descriptor.

This hierarchy is enforced meaning that if you configure a rule which adds a session name descriptor followed by a rule which adds a version descriptor, the SDP will be created with the version descriptor first, followed by the session name.

The only validation that will occur is the prevention of adding duplicate values. In much the same way that header-rules prevents the user from adding multiple To headers, the descriptor rule will not allow the user to add multiple descriptors; unless multiple descriptors are allowed, as is in the case of b, t, r and a.

There exists a parameter **type** under the sdp-line-rule object that allows the user to specify the specific line on which to perform the operation. For example: v, o, s, i, u, e, p, c, b, t, r, z, k, a, and m. Details on these types can be found in RFC 4566.

For those descriptors, of which there may exist zero or more (b, t, r, and a) entries, indexing grammar may be used to reference the specific instance of that attribute. This indexing grammar is consistent with that of header-rules for referring to multiple headers of the same type.

Given the example SDP below:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
r=604800 3600 0 90000
r=7d 1h 0 25h
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
a=orient:portrait
```

and the following sdp-line-rule:

```
sdp-line-rule
   name                 removeRepeatInterval
   type                 r[1]
   action               delete
```

The rule removeRepeatInterval removes the second repeat interval descriptor within the SDP.

The behavior of all SDP rules follow the same behavior of all manipulation rules in that they are executed in the order in which they are configured and that each rule executes on the resultant of the previous rule.

Each descriptor follows its own grammar and rules depending on the type specified. The values of the descriptor are evaluated at runtime since the new-values themselves are evaluated at runtime. At this time no validation of the grammar for each of the types is performed. The user is responsible for properly formatting each of the descriptors according to their specifications.

For instance, the version (v) descriptor can be removed from the SDP but leaving all descriptors for that SDP, causing the SDP to become invalid. This is consistent with the way header-rules operate, in that there is no validation for the specific headers once they have been manipulated through HMR.

Regular Expression Interpolation

An interpolated regular expression is a regular expression that is compiled and evaluated at runtime. Today all regular expressions are compiled at configuration time in order to improve performance. There are cases where a regular expression is determined dynamically from data within a SIP message. In these circumstances the regular expression is unknown until the time of execution.

In order to have a regular expression be interpolated at runtime, it must be contained within a set of {}. An interpolated expression can have any number of regular expressions and strings appended together. Any characters to the left or right of the curly braces will be appended to the value within the curly braces. The curly braces are effectively two operators treated as one (interpolate the value contained within and then concatenate the values to the left and right of

the curly braces). If the comparison-type is set to pattern-rule and the match-value contains a value that matches the grammar below, then it will be treated as an interpolated expression.

```
([^\\]|^)\{\$[^0-9]+[^}]*\}
```

The example below demonstrates using a user defined variable within a regular expression of another rule at runtime.

```
element-rule

        name                            someRule
        type                            header-value
        action                          replace
        comparison-type                 pattern-rule
        match-value                     ^sip:{$rule1.$0}@(.+)$
        new-value                       sip:bob@example.com
```

If the value of $rule1.$0 evaluates to alice then it will successfully match against the string sip:alice@comcast.net. An interpolated expression can be as simple as “{$rule1.$0}” or as complex as ^sip:{rule1.$0}@{$rule2[1].$2}$. It is not possible to interpolate a normal regular expression since the grammar will not allow the user to enter such an expression. Only variables can be contained with the curly braces.

The resultant of interpolated expressions can be stored in user defined variables. Given the same example from above, if the rule someRule was referenced by another rule, the value of sip:alice@comcast.net would be stored within that rule.

Interpolation only makes a single pass at interpolation, but does so every time the Rule executes. In other words, if the Rule is applied to the Route header, it will interpolate again for each Route header instance. What this means is that the value within the curly braces will only be evaluated once. For instance, if the value {$someRule.$1} evaluates to {$foobar.$2} the SBC will treat $foobar.$2 as a literal string which it will compile as a regular expression. The SBC will not recursively attempt to evaluate $foobar.$2, even if it was a valid user defined variable.

Interpolated regular expressions will evaluate to TRUE if and only if both the regular expression itself can be compiled and it successfully matches against the compared string.

You cannot use both interpolated expressions and number quantifiers like `{3,5}` in the same match-value. When interpolated expressions are evaluated, the brackets around the number quantifiers will be removed, leaving the literal string `3,5`. For example, if `$someRule.$1` resolves to a literal string 101, then a match-value of `^[0-9]{3,5} RTP.* {$someRule.$1}` will resolve to `^[0-9]3,5 RTP.* 101`, which will not match any number 3 to 5 times.

Regular Expressions as Boolean Expressions

Regular expressions can be used as boolean expressions today if they are the only value being compared against a string, as is shown in the case below.

```
mime-rule
        name                            someMimeRule
        content-type                    application/text
        action                          replace
        comparison-type                 pattern-rule
        match-value                     ^every good boy .*
        new-value                       every good girl does fine
```

```
However, regular expressions can not be used in conjunction with other
boolean expressions to form more complex boolean expressions, as is shown
below.

mime-rule
     name                      someMimeRule
     content-type              application/text
     action                    replace
     comparison-type           boolean
     match-value               $someRule & ^every good boy .*
     new-value                 every good girl does fine
```

There are many cases where the user has the need to compare some value as a regular expression in conjunction with another stored value. It is possible to perform this behavior today, however it requires an extra step in first storing the value with the regular expression, followed by another Manipulation Rule which compares the two boolean expressions together (e.g. $someRule & $someMimeRule).

In order to simplify the configuration of some sip-manipulations and to make them more efficient this functionality is being added.

Unfortunately, it is not possible to just use the example as is shown above. The problem is there are many characters that are commonly used in regular expressions that would confuse the HMR expression parser (such as $, and +). Therefore delimiting characters need to be used to separate the regular expression from the other parts of the expression.

To treat a regular expression as a boolean expression, it needs to be enclosed within the value $REGEX(<expression>,<compare_string>=$ORIGINAL); where <expression> is the regular expression to be evaluated. <compare_string> is the string to compare against the regular expression. This second argument to the function is defaulted to $ORIGINAL which is the value of the of the specific Manipulation Rule object. It can be overridden to be any other value the user desires.

The proper configuration for the example above to use regular expressions as boolean expressions is

```
mime-rule
        name              someMimeRule
        content-type      application/text
        action            replace
        comparison-type   boolean
        match-value       $someRule & $REGEX(“^every good boy .*”)
        new-value         every good girl does fine
```

It is also possible to use expressions as arguments to the $REGEX function. These expressions will in turn be evaluated prior to executing the $REGEX function. A more complex example is illustrated below.

```
header-rule
        name               checkPAU
        header-name        request-uri
        action             reject
        comparison-type    boolean
        match-value        (!$REGEX($rule1[0],$FROM_USER))&
                           (!$REGEX($rule2[0],$PAI_USER))
        msg-type           request
```

```
        new-value           403:Forbidden
        methods             INVITE,SUBSCRIBE,MESSAGE,PUBLISH,
                            OPTIONS, REFER
```

It should be noted that when using $REGEX() in a boolean expression, the result of that expression is not stored in the user variable. The comparison-type must be set to pattern-rule in order to store the result of a regular expression.

The arguments to the $REGEX() function are interpolated by default. This is the case since the arguments themselves must be evaluated at runtime. The following example is also valid.

```
mime-rule
        name                  someMimeRule
        content-type          application/text
        action                replace
        comparison-type       boolean
        match-value           $someRule & $REGEX(“^every good
                              {$rule1[0].$0} .*”)
```

Moving Manipulation Rules

You can move rules within any manipulation-rule container. Any manipulation rule that contains sub-rules offers the ACLI command **move** <from index> <to index>. For example, given the order and list of rules below:

1. rule1

2. rule2

3. rule3

4. rule4

You can move rule3 to position 1 by executing **move 3 1**. The resulting order is: rule3, rule1, rule2, rule4. A move operation causes a shift (or insert before) for all other rules. When you move a rule from the top or middle to the bottom, the system shifts all rules above the bottom up to the position of the rule that you moved. When you move a rule from the bottom or middle to the top, the system shifts all rules below down to the position of the rule that you moved. Positions start from 1.

A valid from-index and to-index are required to be supplied as arguments to the move action. If you enter a range that is out of bounds for either the from-index or to-index, the ACLI informs you that the command did not execute and the reason.

If you create an invalid sip-manipulation by incorrectly ordering the manipulation rules, the Oracle Communications Session Border Controller validates the rules at configuration time and treats them as invalid prior to runtime. This may or may not affect the outcome of the sip-manipulation as a configured rule may not perform any operation if it refers to a rule that has yet to be executed. It is your responsibility to reorder the remaining rules in order to make the sip-manipulation valid again.

Note that rules of a different type at the same level are all part of the same list. Header-rules, mime-rules, mime-isup-rules, and mime-sdp-rules all share the same configuration level under sip-manipulation. When selecting a move from-index and to-index for a header-rule, you must take into consideration the location of all other rules at the same level because the move is relative to all rules at that level. The move is not relative to the particular rule you selected (for example, the header-rule).

Because the list of rules at any one level can be lengthy, you can issue the **move** command one argument at a time, providing you with the ability to select indices. For example, typing **move** without any arguments displays the list of all the rules at that level. After selecting an appropriate index, the system prompts you with a to-index location based on the same list provided.

For Example:

```
ORACLE(sip-mime-sdp-rules)# move
select a rule to move


1: msr sdp-type=any; action=none; match-value=; msg-type=any

2: addFoo header-name=Foo; action=none; match-value=; msg-type=any

3: addBar header-name=Bar; action=none; match-value=; msg-type=any

selection: 2
destination: 1
Rule moved from position 2 to position 1
ACMEPACKET(sip-mime-sdp-rules)#
```

Rule Nesting and Management

There will be cases where the user wants to take a stored value from the SDP and place it in a SIP header, and vice-versa. All header-rules, element-rules, mime-rules, mime-isup-rules, isup-param-rules, mime-header-rules and mime-sdp-rules are inherited from a Manipulation Rule. A Sip Manipulation is of type Manipulation which contains a list of Manipulation Rules. Each Manipulation Rule can itself contain a list of Manipulation Rules. Therefore when configuring manipulation rules, they will be saved in the order which they have been configured. This is different from the way other configuration objects are configured. Essentially, the user has the option of configuring which type of object they want and when they are done, it gets added to the end of the sip-manipulation, such that order is preserved. This will mean that any Manipulation Rule at the same level can not share the same name. For example, names of header-rules can't be the same as any of the mime-sdp-rule ones or mime-isup-rule. This allows the user to reference stored values from one rule type in another at the same level.

ACLI Configuration Examples

The following eight sections provide sample SDP manipulations.

Remove SDP

```
sip-manipulation
        name                    stripSdp
        description             remove SDP from SIP message
        mime-sdp-rule
                name                    sdpStrip
                msg-type                request
                methods                 INVITE
                action                  delete
                comparison-type         case-sensitive
```

```
                match-value
                new-value
```

Remove Video from SDP

```
sip-manipulation
        name                        stripVideo
        description                 strip video codecs from SIP
                                    message
        mime-sdp-rule
                name                   stripVideo
                msg-type               request
                methods                INVITE
                action                 manipulate
                comparison-type        case-sensitive
                match-value
                new-value
                sdp-media-rule
                        name              removeVideo
                        media-type        video
                        action            delete
                        comparison-type   case-sensitive
match-value
                        new-value
```

Add SDP

```
sip-manipulation
        name                      addSdp
        description               add an entire SDP if one does
                                  not exist
        mime-sdp-rule
                name                   addSdp
                msg-type               request
                methods                INVITE
                action                 add
                comparison-type        case-sensitive
                match-value
                new-value              “v=0\r\no=mhandley
2890844526 2890842807 IN IP4 “+$LOCAL_IP+”\r\ns=SDP Seminar\r\ni=A
Seminar on the session description protocol\r\nu=http:
//www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps\r\ne=mjh@isi.edu
(Mark Handley)\r\nc=IN IP4 “+$LOCAL_IP+”\r\nt=2873397496
2873404696\r\na=recvonly\r\nm=audio 49170 RTP/AVP 0\r\n”
```

Manipulate Contacts

This rule changes the contact in the SDP to the value contained in the Contact header.

```
sip-manipulation
        name                          changeSdpContact
        description                   changes the contact in the SDP to the
value of the contact header
```

```
                header-rule
                        name                            storeContact
                        header-name                     Contact
                        action                          store
                        comparison-type                 pattern-rule
                        msg-type                        request
                        methods                         INVITE
                        match-value
                        new-value
                        element-rule
                                name                            storeHost
                                parameter-name
                                type                            uri-host
                                action                          store
                                match-val-type                  ip
                                comparison-type                 pattern-rule
                                match-value
                                new-value
                mime-sdp-rule
                        name                            changeConnection
                        msg-type                        request
                        methods                         INVITE
                        action                          manipulate
                        comparison-type                 case-sensitive
                        match-value
                        new-value
                        sdp-session-rule
                                name                    changeCLine
                                action                  manipulate
                                comparison-type         case-sensitive
                                match-value
                                new-value
                                sdp-line-rule
                                   name                 updateConnection
                                   type                 c
                                   action               replace
                                   comparison-type      case-sensitive
                                   match-value          $storeContact.$storeHost
                                   new-value            $storeContact.$storeHost.$0
```

Remove a Codec

This rule changes the contact in the SDP to the value contained in the Contact header.

```
sip-manipulation
        name                            removeCodec
        description                     remove G711 codec if it exists
        mime-sdp-rule
                name                            removeCodec
                msg-type                        request
                methods                         INVITE
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                new-value
```

```
                    sdp-media-rule
                            name                        removeG711
                            media-type                  audio
                            action                      manipulate
                            comparison-type             case-sensitive
                            match-value
                            new-value
                            sdp-line-rule
                                    name                    remove711
                                    type                    m
                                    action                  replace
                                    comparison-type         pattern-rule
                                    match-value             ^(audio [0-9]
                                                            {1,5} RTP.*)( [07]
                                                            \b)(.*)$
                                    new-value               $1+$3
                            sdp-line-rule
                                    name                    stripAttr
                                    type                    a
                                    action                  delete
                                    comparison-type         pattern-rule
                                    match-value             ^(rtpmap|fmtp):
                                                            [07]\b$
                                    new-value
```

Change Codec

```
            sip-manipulation
                    name                            convertCodec
                    description                     changeG711toG729
                    mime-sdp-rule
                            name                            changeCodec
                            msg-type                        request
                            methods                         INVITE
                            action                          manipulate
                            comparison-type                 case-sensitive
                            match-value
                            new-value
                            sdp-media-rule
                                    name                      change711to729
                                    media-type                audio
                                    action                    manipulate
                                    comparison-type           case-sensitive
                                    match-value
                                    new-value
                                    sdp-line-rule
                                            name                  change711
                                            type                  m
                                            action                replace
                                            comparison-type       pattern-rule
                                            match-value           ^(audio [0-9]{4,5}
                                                                  RTP/AVP.*)( 0)(.*)$
            new-value             $1+" 18"+$3
                                    sdp-line-rule
                                            name                  stripAttr
```

```
                                type                 a
                                action               delete
                                comparison-type      pattern-rule
                                match-value          ^rtpmap:0 PCMU/
                                                     .+$
                                new-value
                        sdp-line-rule
                                name                 addAttr
                                type                 a
                                action               add
                                comparison-type      boolean
                                match-value          $change711to729.
                                                     $stripAttr
                                new-value            rtpmap:18 G729/8000
```

Remove Last Codec and Change Port

```
sip-manipulation
        name                            removeLastCodec
        description                     remove the last codec
        mime-sdp-rule
                name                            removeLastCodec
                msg-type                        request
                methods                         INVITE
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                new-value
                sdp-media-rule
                        name                        removeLast
                        media-type                  audio
                        action                      manipulate
                        comparison-type             case-sensitive
                        match-value
                        new-value
                        sdp-line-rule
                                name                    isLastCodec
                                type                    m
                                action                  store
                                comparison-type         pattern-rule
                                match-value             ^(audio )([0-9]{4,
                                                        5})( RTP/AVP
                                                        [0-9]{1-3})$
new-value
                        sdp-line-rule
                                name                    changePort
                                type                    m
                                action                  replace
                                comparison-type         boolean
                                match-value             $removeLastCodec.
$removeLast.$isLastCodec
                                new-value               $removeLastCodec.
$removeLast.$isLastCodec.$1+0+$removeLastCodec.$removeLast.
$isLastCodec.$3
```

Remove Codec with Dynamic Payload

```
sip-manipulation
    name                        removeAMR
    description                 remove the AMR and AMR-WB dynamic codecs
    split-headers
    join-headers
    mime-sdp-rule
        name                        sdpAMR
        msg-type                    request
        methods                     INVITE
        action                      manipulate
        comparison-type             case-sensitive
        match-value
        new-value
        sdp-media-rule
            name                        mediaAMR
            media-type                  audio
            action                      manipulate
            comparison-type             case-sensitive
            match-value
            new-value
            sdp-line-rule
                name                        isAMR
                type                        a
                action                      delete
                comparison-type             pattern-rule
                match-value                 ^rtpmap:([0-9]{2,3}) AMR\/
                new-value
        sdp-media-rule
            name                        mediaIsAMR
            media-type                  audio
            action                      manipulate
            comparison-type             boolean
            match-value                 $sdpAMR.$mediaAMR.$isAMR[~]
            new-value
            sdp-line-rule
                name                        delFmtpAMR
                type                        a
                action                      delete
                comparison-type             pattern-rule
                match-value                 ^fmtp:
({$sdpAMR.$mediaAMR.$isAMR[~].$1})
                new-value
            sdp-line-rule
                name                        delAMRcodec
                type                        m
                action                      find-replace-all
                comparison-type             pattern-rule
                match-value                 ^(audio [0-9]+ RTP.*)
{$sdpAMR.$mediaAMR.$isAMR[~].$1}(.*)$
                new-value                   $1+$2
```

HMR Import-Export

Due to the complexity of SIP manipulations rules and the deep understanding of system syntax they require, it is often difficult to configure reliable rules. This feature provides support for importing and exporting pieces of SIP manipulation configuration in a reliable way so that they can be reused.

Exporting

The SIP manipulation configuration contains an **export** command which sends the previously selected configuration to the designated file. The syntax is **export [FILENAME]**. The system compresses the file with gzip and writes it to the `/code/imports` directory.

> **Note**
>
> SIP manipulation configurations can only be exported one at a time.

Exported data will look like this:

```
<?xml version='1.0' standalone='yes'?>
<sipManipulation
        name='manip'
        description=''
        lastModifiedBy='admin@console'
        lastModifiedDate='2009-10-16 14:16:29'>
        <headerRule
                headerName='Foo'
                msgType='any'
                name='headerRule'
                action='manipulate'
                cmpType='boolean'
                matchValue='$REGEX("[bB][A-Za-z]{2}")'
                newValue='foo'
                methods='INVITE'>
        </headerRule>
</sipManipulation>
```

To avoid conflicts when importing, the key and object ID are not included as part of the exported XML.

Importing

The **import** command imports data from a previously exported file into the currently-selected configuration. If no configuration was selected, a new one is created. The syntax is **import [FILENAME]**. Include the .gz extension in the filename. After importing, type **done** to save the configuration.

Importing a configuration with the same key as one that already exists returns an error. In this case:

- Delete the object with the same key and re-import.

- Select the object with the same key and perform an import that will overwrite it with new data.

Using SFTP to Move Files

After exporting a configuration, use SFTP to copy the file to other Oracle Communications Session Border Controllers. Place the file in the `/code/imports` directory before using the **import** command on the second SBC.

Removing Files

Using the **delete-import** command with the name of the file you want to delete removes it from the system. Using this command, you can delete files that are no longer useful to you. Carrying out this command is final and there is no warning before you go ahead with the deletion. A failed deletion (for instance, because there is no such file) will produce an error message; a successful deletion simply returns you to the system prompt.

HMR Development

Before you start developing an HMR, ask yourself whether you need an HMR. Check whether an alternative is available. For example, you can configure the SBC to perform some of the more common needed message manipulations like stripping telephone events from SDP or resolving delayed offer issues. If you need more flexibility to address your problem, then HMR is probably the answer.

Development Overview

Once you have decided you want to use HMR to resolve an issue, Oracle recommends you follow this development procedure:

1. Understand regex. Your knowledge of regex is fundamental to building an HMR that yields the desired result.
2. Identify the direction of the traffic in relation to the SBC to which you want to apply an HMR (inbound or outbound).
3. Identify the SIP message portion on which you want the HMR to operate: header, parameter, or body.
4. Identify the remote entities involved and know their represented in your SBC configuration. Are they session agents, realms or SIP interfaces? Take into consideration the order of precedence among these entities for applying HMRs.
5. Build the HMR and test it using the SBC's Testing SIP Manipulations.
6. Apply the HMR appropriately to your configuration. Oracle recommends that you develop, test, and apply HMRs in test or laboratory environments only.
7. Analyze the data resulting from your HMR to confirm it is working as you intend.

Development Tips

- Define all storage rules first. Each subsequent header rule processes against the same SIP message, so each additional header rules works off of the results from the application of the rule that precedes it.
  In general, you want to store values from the original SIP header rather than from the iteratively changed versions.

- Implement rules at the element rule rather than the header rule level. Header rules should only be a container for element rules.
- Add additional element rules to modify a single header. Do not create multiple header rules, each with one element rule. Instead, create multiple element rules within a header rule.
- Think of performance. Reuse as many built in variables as possible
- Avoid lengthy string matches unless absolutely necessary
- Wherever possible, constrain your HMR appropriately by specifying a SIP method and message type
- Build an HMR library

Planning Considerations

You want to plan your functionality carefully when developing HMRs and you want to test it thoroughly before deploying it on your production system.

Traffic Direction

You need to determine if you want changes to occur on traffic that is relative to the SBC inbound or outbound.

Order of Application Precedence

As you decide direction, you must also consider the order in which the SBC applies HMR for session agents, realms, and SIP interfaces. The order of precedence is:

- session agent
- realm
- SIP interface

A SIP manipulation applied to a session agent overrides the other two, and a SIP manipulation for a realm overrides one for a SIP interface.

Order of HMR Execution

The SBC applies SIP header rules in the order you have entered them, which guards against the removal of data that might be used by other header rules. The order starts with the top-most rule and continues with the execution of the sub-rules one by one. Each new rule is carried out on the result of the preceding rule.

This ordering also lets you strategically use manipulations. For example, you can use two rules if you want to store the values of a regular expression. The first rule stores the value of a matched regular expression and the second deletes the matched value.

Applying HMR to a Specific Header

You can operate on a specific instance of a given header by adding a trailing `[<index>]` value after the header name. This `[<index>]` is a numerical value representing the specific instance of the header on which to operate. However, the SBC takes no action if the header does not exist. You can also use the caret `^` to reference the last header of that type if there are multiple instances.

The count for referencing is zero-based, meaning that the first instance of the header counts as 0.

> **Note**
>
> You cannot use a trailing `[<index>]` value after the header name to insert headers into a specific location. Headers are added to the end of the list, except that Via headers are added to the top.

HMR Sets

Although the SBC has a set method for how certain manipulation rules take precedence over others; you can use multiple SIP HMR sets to

- Apply multiple inbound and outbound manipulations rules to a SIP message
- Provision the order in which the SBC applies HMRs

You cause the header rule in one HMR to invoke another HMR. Values from that invoked HMR for the match value, comparison type, and methods are then supported. The invoked HMR is performed when those values are true.

Create Pseudocode

You start with a high-level design, refine the design to pseudocode, and then refine the pseudocode to source code. This successive refinement in small steps allows you to check your design as you drive it to lower levels of detail. The result is that you catch high level errors at the highest level, mid-level errors at the middle level, and low-level errors at the lowest level -- before any of them becomes a problem or contaminates work at more detailed levels.

Test HMRs

Test methodologies include:

- Wireshark traces to create SIPp scripts
- test-pattern-rule to test pattern matches from the ACLI
- test-sip-manipulation available through the ACLI
- log.sipd messages

test-sip-manipulation

You can use a tool that allows you to test the outcome of your SIP manipulation and header rules without sending real traffic through the SBC to see if they work.

To use the tool, you enter the ACLI's test-sip-manipulation utility and reference the rule you want to test using its name. Then you enter a mode where you put in a SIP message entered in ASCII. You can cut and paste this message from sipmsg.log or from some other location. Using <Ctrl-D> stops the SIP message collection and parses it.

The test informs you of any parsing errors found in the SIP message. Once the message is entered, you can execute the SIP manipulation against the message. The output after this step is the modified SIP message after manipulations have been applied. You will also find a debugging option, which displays SIP manipulation logging to the screen as the manipulation takes place.

As a starting point for testing, this tool comes loaded with a default SIP message. It cannot be associated with realms, session agents, or SIP interfaces, and so it also comes with certain resolves reserved words, such as: `$LOCAL_IP`, `$TRUNK_GROUP_CONTEXT`, and `$REMOTE_PORT`. In addition, you can use your settings for testing across terminal sessions; if you choose to save your settings, everything (including the SIP message) will be saved, with the exception of the debugging option.

It is not recommended that you use this tool to add an ISUP message body.

Development Example

You want to perform specialized call routing for x11 numbers, such as 211, 311, 411 and so on, based on from where the call originated. You want to concatenate the user part of the To URI with the seven digits following the +1 in the user part of the From URI and to swap that value in the user part of the Request URI:

```
INVITE sip:211;csel=nonind@192.168.65.16:5060;user=phone SIP/2.0
Via:SIP/2.0/UDP 10.1.110.34;branch=z9hG4bK-
BroadWorks.as3.otwaon10-192.168.65.16V5060-0-31288454-509069652-1273520380170-
From:"JOHN SMITH"<sip:+14167601262@sipt.itech.ca;user=phone>
To:<sip:211;csel=noind@92.168.65.16:5060;user=phone>
```

Note

- To user-uri: 211
- From user-uri: +14167601262
- Desired Request-URI: 2114167601

Writing the Pseudo Code

- Header rule getToURI for To header is not needed. The built-in variable $RURI_USER can be used.
- Header rule getFromURIDigits for From header. Stores specific digits for the uri-user-only part of the From header.
- Header rule constructRURIUsingToAndFrom to build the Request-URI. Replaces the uri-user of the Request-URI with a concatenation of the stored digits.

Testing the Pattern Rule

```
(configure)# session-router test-pattern-rule
(test-pattern-rule)# string +14167601262
expression made 0 matches against string
(test-pattern-rule)# expression ^\+1([0-9]{7}).*$
expression made 2 matches again string
(test-pattern-rule)# show
Pattern Rule:
Expression    : ^\+1([0-9]{7}).*$
String    : +14167601262
Matched    : TRUE
Matches:
```

```
$0 +14167601262
$1 4167601
```

> **Note**
>
> - A $ was used to denote the end of the string. Using a carriage return line feed \r\n will not result in matches.
> - $0 is the entire string being matched against.
> - $1 is the string represented in the first set of parentheses. Here, $1 matches the desired output so the regular expression is correct.

Constructing the HMR

```
sip-manipulation
name                        ContructURI
description
header-rule
    name                    getFromURIDigits
    header-name             From
    action                  store
    comparison-type         case-sensitive
    match-value
    msg-type                request
    new-value
    methods                 INVITE
    element-rule
        name                getDigit
        parameter-name
        type                uri-user
        action              store
        match-val-type      any
        comparison-type     pattern-rule
        match-value         ^\+1([0-9]{7}).*$
        new-value
header-rule
    name                    constructRURIUsingToAndFrom
    header-name             request-uri
    action                  manipulate
    comparison-type         case-insensitive
    match-value
    msg-type                request
    new-value
    methods                 INVITE
    element-rule
        name                constructRURI
        parameter-name
        type                uri-user
        action              replace
        match-val-type      any
        comparison-type     pattern-rule
```

```
match-value
new-value           $RURI_USER.$0+$getFromURIDigits.$getDigits.$1
```

> **Note**
>
> `$RURI_USER.$0+$getFromURIDigits.$getDigits.$1`
> Concatenate the two and replace the uri-user of the R-URI. The plus sign (+) serves as the concatenation operator when the comparison-type is pattern-rule. Only the $1 from the second ruleset is used because it represents just the subset of the From digits needed.

Loading Test SIP Message

```
(test-sip-manipulation)# load-sip-message
```

You might want to edit the Content-Length value default value of 276 or to remove the header. Retaining that value causes test-sip-manipulation to transmit only the first 276 characters of the loaded SIP message.

Configuring Testing

```
Test Sip Manipulation:
    sip-manipulation           : ConstructRURI
    debugging                  : enabled
    direction                  : out
    manipulation-string        :
    manipulation-pattern       : \,+
    tgrp-context               :
    local-ip                   : 192.168.1.60:5060
    remote-ip                  : 192.168.1.61:5060
    sip-message                : parsed OK
```

Executing Testing

```
(test-sip-manipulation)# execute
Header Rule ConstructRURI (headerName=request-uri action=manipulate
cmpType=pattern-rule) does not apply to method INVITE
After Manipulation[ConstructRURI]
```

The following output snippet shows that the HMR worked:

```
INVITE sip:2114167601@192.168.65.16:5060;user=phone SIP/2.0
Via: SIP/2.0/UDP 10.1.119.152:5060;branch=x9hG4bKj3svpd1030b08nc9t3f1.1
From: JOHN SMITH<sip:
+14167601262@sipt.tech.ca;user=phone;tag=SDekcfd01-966714349-1273696750280-
To: <sip:211;csel=noind@10.1.119.151:5060;user=phone
```

Log File Analysis

Run log.sipd at debug level on the SBC where you plan to test the HMR to gain the most information. Then examine log.sipd to review information about the HMR execution.

Configuration Examples

This section shows you several configuration examples for HMR. This section shows the configuration for the various rules that the Oracle Communications Session Border Controllerapplied, and sample results of the manipulation. These examples present configurations as an entire list of fields and settings for each ruleset, nested header rules and nested element rules. If a field does not have any operation within the set, the field is shown with the setting at the default or blank.

Example 1 Removing Headers

For this manipulation rule, the Oracle Communications Session Border Controller removes the Custom header if it matches the pattern rule. It stores the defined pattern rule for the goodBye header. Finally, it removes the goodBye header if the pattern rule from above is a match.

This is a sample of the configuration:

```
sip-manipulation
        name                            removeHeader
        header-rule
                name                            removeCustom
                header-name                     Custom
                action                          delete
                comparison-type                 boolean
                match-value                     ^This is my.*
                msg-type                        request
                new-value
                methods                         INVITE
        header-rule
                name                            goodByeHeader
                header-name                     Goodbye
                action                          store
                comparison-type                 boolean
                match-value                     ^Remove (.+)
                msg-type                        request
                new-value
                methods                         INVITE
        header-rule
                name                            goodBye
                action                          delete
                comparison-type                 pattern-rule
                match-value                     $goodByeHeader
                msg-type                        request
                new-value
                methods                         INVITE
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK0g639r10fgc0aakk26s1.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SDc1rm601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SDc1rm601-d01673bcacfcc112c053d95971330335-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 2 Manipulating the Request URI

For this manipulation rules, the Oracle Communications Session Border Controller stores the URI parameter tgid in the Request URI. Then if the pattern rule matches, it adds a new header (x-customer-profile) with the a new header value tgid to the URI parameter in the request URI.

This is a sample of the configuration:

```
sip-manipulation
        name                            CustomerTgid
        header-rule
                name                            ruriRegex
                header-name                     request-uri
                action                          store
                comparison-type                 pattern-rule
                match-value
                msg-type                        request
new-value
                methods                         INVITE
                element-rule
                        name                            tgidParam
                        parameter-name                  tgid
                        type                            uri-param
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
header-rule
                name                            addCustomer
                header-name                     X-Customer-Profile
                action                          add
                comparison-type                 pattern-rule
                match-value                     $ruriRegex.$tgidParam
                msg-type                        request
```

```
                new-value                       $ruriRegex.$tgidParam.$0
                methods                         INVITE
header-rule
                name                            delTgid
                header-name                     request-uri
                action                          manipulate
                comparison-type                 pattern-rule
                match-value                     $ruriRegex.$tgidParam
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            tgidParam
                        parameter-name                  tgid
                        type                            uri-param
                        action                          delete-element
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-
value                    $ruriRegex.$tgidParam.$0
                        new-value
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060 SIP/2.0
    Message Header
Via: SIP/2.0/UDP 192.168.200.61:5060;branch=z9hG4bK0g6plv3088h03acgh6c1.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SDc1rg601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SDc1rg601-f125d8b0ec7985c378b04cab9f91cc09-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
        X-Customer-Profile: 123
```

Example 3 Manipulating a Header

For this manipulation rule, the Oracle Communications Session Border Controllerstores the pattern matches for the Custom header, and replaces the value of the Custom header with a combination of the stored matches and new content.

This is a sample of the configuration:

```
sip-manipulation
        name                            modCustomHdr
        header-rule
```

```
                name                            customSearch
                header-name                     Custom
                action                          store
                comparison-type                 pattern-rule
                match-value                     (This is my )(.+)( header)
                msg-type                        request
                new-value
                methods                         INVITE
header-rule
                name                            customMod
                header-name                     Custom
                action                          manipulate
                comparison-type                 pattern-rule
                match-value                     $customSearch
                msg-type                        request
                new-value
methods                         INVITE
                element-rule
                        name                            hdrVal
                        parameter-name                  hdrVal
                        type                            header-value
                        action                          replace
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
new-value                       $customSearch.$1+edited+$customSearch.$3
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK20q2s820boghbacgs6o0.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SDe1ra601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SDe1ra601-4bb668e7ec9eeb92c783c78fd5b26586-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my edited header
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 4 Storing and Using URI Parameters

For this manipulation rule, the Oracle Communications Session Border Controller stores the value of the URI parameter tag from the From header. It also creates a new header FromTag with the header value from the stored information resulting from the first rule.

This is a sample of the configuration:

```
sip-manipulation
        name                            storeElemParam
        header-rule
                name                            Frohmr
                header-name                     From
                action                          store
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            elementRule
                        parameter-name                  tag
                        type                            uri-param
                        action                          store
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-value
header-rule
                name                            newHeader
                header-name                     FromTag
                action                          add
                comparison-type                 pattern-rule
                match-value                     $FromHR.$elementRule
                msg-type                        any
                new-value                       $FromHR.$elementRule.$0
                methods
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK4oda2e2050ih7acgh6c1.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SDf1re601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SDf1re601-f85059e74e1b443499587dd2dee504c2-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
Content-Length: 140
        FromTag: 1
```

Example 5 Manipulating Display Names

For this manipulation rule, the Oracle Communications Session Border Controller sores the display name from the Display header. It replaces the two middle characters of the original display name with a new string. Then is also replaces the From header's display name with "abc 123" if it matches sipp.

This is a sample of the configuration:

```
sip-manipulation
        name                            modDisplayParam
        header-rule
                name                            storeDisplay
                header-name                     Display
                action                          store
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            displayName
                        parameter-name                  display
                        type                            uri-display
                        action                          store
                        match-val-type                  any
comparison-type                 pattern-rule
                        match-value                     (s)(ip)(p )
                        new-value
header-rule
                name                            modDisplay
                header-name                     Display
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            modRule
                        parameter-name                  display
                        type                            uri-display
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-
value                    $storeDisplay.$displayName
                        new-
value                      $storeDisplay.$displayName.$1+lur+$storeDisplay.$di
splayName.$3
header-rule
                name                            modFrom
                header-name                     From
                action                          manipulate
                comparison-type                 pattern-rule
```

```
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            fromDisplay
                        parameter-name
                        type                            uri-display
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     sipp
                        new-value                       "\"abc 123\" "
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK681kot109gp04acgs6o0.1
        From: "abc 123" <sip:sipp@192.168.1.60:5060>;tag=SD79ra601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SD79ra601-a487f1259e2370d3dbb558c742d3f8c4-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: slurp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 6 Manipulating Element Parameters

For this more complex manipulation rule, the Oracle Communications Session Border Controller:

- From the Display header, stores the display name, user name, URI parameter up, and header parameter hp
- Adds the header parameter display to the Params header, with the stored value of the display name from the first step
- Add the URI parameter user to the Params header, with the stored value of the display name from the first step
- If the URI parameter match succeeds in the first step, replaces the URI parameter up with the Display header with the value def
- If the header parameter match succeeds in the first step, deletes the header parameter hp from the Display header

This is a sample of the configuration:

```
sip-manipulation
        name                            elemParams
        header-rule
                name                            StoreDisplay
                header-name                     Display
                action                          store
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            displayName
                        parameter-name
                        type                            uri-display
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
element-rule
                        name                            userName
                        parameter-name                  user
                        type                            uri-user
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
element-rule
                        name                            uriParam
                        parameter-name                  up
                        type                            uri-param
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
element-rule
                        name                            headerParam
                        parameter-name                  hp
                        type                            header-param
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
        header-rule
                name                            EditParams
                header-name                     Params
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
```

```
                new-value
                methods                         INVITE
                element-rule
                        name                            addHeaderParam
                        parameter-name                  display
                        type                            header-param
                        action                          add
match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-
value                      $StoreDisplay.$displayName.$0
                element-rule
                        name                            addUriParam
                        parameter-name                  user
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
               new-value
$StoreDisplay.$userName.$0
        header-rule
                name                            EditDisplay
                header-name                     Display
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            replaceUriParam
                        parameter-name                  up
                        type                            uri-param
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     $StoreDisplay.$uriParam
                        new-value                       def
                element-rule
                        name                            delHeaderParam
                        parameter-name                  hp
                        type                            header-param
                        action                          delete-element
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value $StoreDisplay.$headerParam
                        new-value
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
```

```
192.168.200.61:5060;branch=z9hG4bK7okvei0028jgdacgh6c1.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SD89rm601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SD89rm601-b5b746cef19d0154cb1f342cb04ec3cb-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: sipp <sip:user@192.168.1.60:5060;up=def>
        Params: sipp <sip:sipp1@192.168.1.60:5060;user=user>;display=sipp
        Params: sipp <sip:sipp2@192.168.1.60:5060;user=user>;display=sipp
        Edit: disp <sip:user@192.168.1.60:5060>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 7 Accessing Data from Multiple Headers of the Same Type

For this manipulation rule, the Oracle Communications Session Border Controller stores the user name from the Params header. It then adds the URI parameter c1 with the value stored from the first Params header. Finally, it adds the URI parameter c2 with the value stored from the second Params header.

This is a sample of the configuration:

```
sip-manipulation
        name                            Params
        header-rule
                name                            storeParams
                header-name                     Params
                action                          store
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            storeUserName
                        parameter-name                  user
                        type                            uri-user
                        action                          store
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-value
header-rule
                name                            modEdit
                header-name                     Edit
                action                          manipulate
                comparison-type                 pattern-rule
                match-value
                msg-type                        request
                new-value
methods                         INVITE
```

```
                element-rule
                        name                            addParam1
                        parameter-name                  c1
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-
value                      $storeParams[0].$storeUserName.$0
                element-rule
                        name                            addParam2
                        parameter-name                  c2
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-
value                      $storeParams[1].$storeUserName.$0
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK9g855p30cos08acgs6o0.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SD99ri601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SD99ri601-6f5691f6461356f607b0737e4039caec-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp <sip:user@192.168.1.60:5060;c1=sipp1;c2=sipp2>
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 8 Using Header Rule Special Characters

For this manipulation rule, the Oracle Communications Session Border Controller:

- Stores the header value of the Params header with the given pattern rule, and stores both the user name of the Params header and the URI parameter abc
- Adds the URI parameter lpu with the value stored from the previous Params header
- If any of the Params headers match the pattern rule defined in the first step, adds the URI parameter apu with the value aup

- If all of the Params headers match the pattern rule defined in the first step, adds the URI parameter apu with the value apu
- If the first Params headers does not match the pattern rule for storing the URI parameter defined in the first step, adds the URI parameter not with the value 123
- If the first Params headers matches the pattern rule for storing the URI parameter defined in the first step, adds the URI parameter yes with the value 456

This is a sample of the configuration:

```
sip-manipulation
        name                            specialChar
        header-rule
                name                            searchParams
                header-name                     Params
                action                          store
                comparison-type                 pattern-rule
                match-value                     .*sip:(.+)@.*
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            userName
                        parameter-name
                        type                            uri-user
                        action                          store
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
                        new-value
element-rule
                        name                            emptyUriParam
                        parameter-name                  abc
                        type                            uri-param
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value
                        new-value
header-rule
                name                            addUserLast
                header-name                     Edit
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            lastParamUser
                        parameter-name                  lpu
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value
```

```
                        new-value $searchParams[^].$userName.$0
                element-rule
                        name                            anyParamUser
                        parameter-name                  apu
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     $searchParams[~]
                        new-value                       aup
                element-rule
                        name                            allParamUser
                        parameter-name                  apu
                        type                            header-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     $searchParams[*]
                        new-value                       apu
                element-rule
                        name                            notParamYes
                        parameter-name                  not
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-
value                    !$searchParams.$emptyUriParam
                        new-value                       123
                element-rule
                        name                            notParamNo
                        parameter-name                  yes
                        type                            uri-param
                        action                          add
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-
value                    $searchParams.$emptyUriParam
                        new-value                       456
```

This is a sample of the result:

```
Request-Line: INVITE sip:service@192.168.200.60:5060;tgid=123 SIP/2.0
    Message Header
        Via: SIP/2.0/UDP
192.168.200.61:5060;branch=z9hG4bK681m9t30e0qh6akgj2s1.1
        From: sipp <sip:sipp@192.168.1.60:5060>;tag=SDchrc601-1
        To: sut <sip:service@192.168.1.61:5060>
        Call-ID: SDchrc601-fcf5660a56e2131fd27f12fcbd169fe8-06a3gu0
        CSeq: 1 INVITE
        Contact: <sip:sipp@192.168.200.61:5060;transport=udp>
        Goodbye: Remove Me
        Custom: This is my custom header
        Display: sipp <sip:user@192.168.1.60:5060;up=abc>;hp=123
        Params: sipp <sip:sipp1@192.168.1.60:5060>
```

```
        Params: sipp <sip:sipp2@192.168.1.60:5060>
        Edit: disp
<sip:user@192.168.1.60:5060;lpu=sipp2;apu=aup;not=123>;apu=apu
        Max-Forwards: 69
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: 140
```

Example 9 Status-Line Manipulation

This section shows an HMR configuration set up for status-line manipulation.

Given that the object of this example is to drop the 183 Session Progress response when it does not have SDP, your SIP manipulation configuration needs to:

1. Search for the 183 Session Progress response
2. Determine if the identified 183 Session Progress responses contain SDP; the Oracle Communications Session Border Controller searches the 183 Session Progress responses where the content length is zero
3. If the 183 Session Progress response does not contain SDP, change its status code to 699
4. Drop all 699 responses

```
sip-manipulation
        name                            manip
        description
        header-rule
                name                            IsContentLength0
                header-name                     Content-Length
                action                          store
                comparison-type                 pattern-rule
                match-value                     0
                msg-type                        reply
                new-value
                methods
        header-rule
                name                            is183
                header-name                     @status-line
                action                          store
                comparison-type                 pattern-rule
                match-value
                msg-type                        reply
                new-value
                methods
                element-rule
name                            is183Code
                        parameter-name
                        type                            status-code
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     183
                        new-value
        header-rule
                name                            change183
```

```
                header-name                     @status-line
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        reply
                new-value
                methods
                element-rule
                        name                            make199
                        parameter-name
                        type                            status-code
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     $IsContentLength0
& $is183.$is183Code
                        new-value                       199

sip-interface     options dropResponse=699
```

Example 10 Use of SIP HMR Sets

The following example shows the configuration for SIP HMR with one SIP manipulation configuration loading another SIP manipulation configuration. The goals of this configuration are to:

- Add a new header to an INVITE
- Store the user portion of the Request URI
- Remove all Route headers from the message only if the Request URI is from a specific user

```
sip-manipulation
        name                            deleteRoute
        description                     delete all Route Headers
        header-rule
                name                            deleteRoute
                header-name                     Route
                action                          delete
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
sip-manipulation
        name                            addAndDelete
        description                     Add a New header and delete Route
headers
        header-rule
                name                            addHeader
                header-name                     New
                action                          add
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
```

```
        new-value                       "Some Value"
        methods                         INVITE
    header-rule
        name                            storeRURI
        header-name                     request-uri
        action                          store
        comparison-type                 pattern-rule
        match-value
        msg-type                        request
        new-value
        methods                         INVITE
        element-rule
            name                            storeUser
            parameter-name
            type                            uri-user
            action                          store
            match-val-type                  any
            comparison-type                 pattern-rule
            match-value                     305.*
            new-value
    header-rule
        name                            deleteHeader
        header-name                     request-uri
        action                          sip-manip
        comparison-type                 Boolean
        match-value                     $storeRURI.$storeUser
        msg-type                        request
        new-value                       deleteRoute
        methods                         INVITE
```

Example 11 Use of Remote and Local Port Information

The following example shows the configuration for remote and local port information. The goals of this configuration are to:

- Add LOCAL_PORT as a header parameter to the From header
- Add REMOTE_PORT as a header parameter to the From header

```
sip-manipulation
        name                            addOrigIp
        description
        header-rule
            name                            addIpParam
            header-name                     From
            action                          manipulate
            comparison-type                 case-sensitive
            match-value
            msg-type                        request
            new-value
            methods                         INVITE
            element-rule
                name                            addIpParam
                parameter-name                  newParam
                type                            header-param
                action                          add
```

```
                match-val-type                  any
                comparison-type                 case-sensitive
                match-value
                new-value                       $LOCAL_IP
        element-rule
                name                            addLocalPort
                parameter-name                  lport
                type                            header-param
                action                          add
                match-val-type                  any
                comparison-type                 case-sensitive
                match-value
                new-value                       $LOCAL_PORT
        element-rule
                name                            addRemotePort
                parameter-name                  rport
                type                            header-param
                action                          add
                match-val-type                  any
                comparison-type                 case-sensitive
                match-value
                new-value                       $REMOTE_PORT
```

Example 12 Response Status Processing

Given that the object of this example is to drop the 183 Session Progress response when it does not have SDP, your SIP manipulation configuration needs to:

1. Search for the 183 Session Progress response
2. Determine if the identified 183 Session Progress responses contain SDP; the Oracle Communications Session Border Controller searches the 183 Session Progress responses where the content length is zero
3. If the 183 Session Progress response does not contain SDP, change its status code to 699
4. Drop all 699 responses

```
sip-manipulation
        name                            manip
        description
        header-rule
                name                            IsContentLength0
                header-name                     Content-Length
                action                          store
                comparison-type                 pattern-rule
                match-value                     0
                msg-type                        reply
                new-value
                methods
        header-rule
                name                            is183
                header-name                     @status-line
                action                          store
                comparison-type                 pattern-rule
                match-value
                msg-type                        reply
```

```
                new-value
                methods
                element-rule
                        name                            is183Code
                        parameter-name
                        type                            status-code
                        action                          store
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     183
                        new-value
        header-rule
                name                            change183
                header-name                     @status-line
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        reply
                new-value
                methods
                element-rule
                        name                            make699
                        parameter-name
                        type                            status-code
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     $IsContentLength0
& $is183.$is183Code
                        new-value                       699
sip-interface
    options dropResponse=699
```

The following four configuration examples are based on the this sample SIP INVITE:

```
INVITE sip:service@192.168.1.61:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.60:5060;branch=z9hG4bK-1-0
From: sipp <sip:sipp@192.168.1.60:5060>;tag=1
To: sut <sip:service@192.168.1.61:5060>
Call-ID: 1-15554@192.168.1.60
CSeq: 1 INVITE
Contact: <sip:sipp@192.168.1.60:5060;user=phone>
Max-Forwards: 70
Content-Type: multipart/mixed;boundary=boundary
Content-Length:  466
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=audio 12345 RTP/AVP 18
a=rtpmap:8 G729/8000/1
a=fmtp:18 annexb=no
```

```
a=sendrecv
a=ptime:20
a=maxptime:200
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=video 12345 RTP/AVP 34
a=rtpmap:34 H263a/90000
a=ptime:30
--boundary--
```

Example 13 Remove a Line from SDP

In this example, the SIP manipulation is configured to remove all p-time attributes from the SDP.

```
sip-manipulation
        name                            removePtimeFromBody
        description                   removes ptime attribute from all bodies
        header-rule
                name                            CTypeManp
                header-name                     Content-Type
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            remPtime
                        parameter-name                  application/sdp
                        type                            mime
                        action                          find-replace-all
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value                     a=ptime:[0-9]{1,2}(\n|
\r\n)
                        new-value
```

The result of manipulating the original SIP INVITE (shown above) with the configured SIP manipulation is:

```
INVITE sip:service@192.168.1.61:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.60:5060;branch=z9hG4bK-1-0
From: sipp <sip:sipp@192.168.1.60:5060>;tag=1
To: sut <sip:service@192.168.1.61:5060>
Call-ID: 1-15554@192.168.1.60
CSeq: 1 INVITE
Contact: <sip:sipp@192.168.1.60:5060;user=phone>
Max-Forwards: 70
```

```
Content-Type: multipart/mixed;boundary=boundary
Content-Length:  466
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=audio 12345 RTP/AVP 18
a=rtpmap:18 G729/8000/1
a=fmtp:18 annexb=no
a=sendrecv
a=maxptime:200
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=video 12345 RTP/AVP 34
a=rtpmap:34 H263a/90000
--boundary-
```

Example 14 Back Reference Syntax

In this sample of back-reference syntax use, the goal is to change the To user. The SIP manipulation would be configured like the following:

```
sip-manipulation
        name                            changeToUser
        description                   change user in the To header
        header-rule
                name                            ChangeHeader
                header-name                     To
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        request
                new-value
                methods                         INVITE
                element-rule
                        name                            replaceValue
                        parameter-name
                        type                            header-value
                        action                          replace
                        match-val-type                  any
                        comparison-type                 pattern-rule
                        match-value                     (.+)(service)(.+)
                        new-value                       $1+Bob+$3
```

The result of manipulating the original SIP INVITE (shown above) with the configured SIP manipulation is:

```
INVITE sip:service@192.168.1.61:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.60:5060;branch=z9hG4bK-1-0
From: sipp <sip:sipp@192.168.1.60:5060>;tag=1
To: sut <sip:Bob@192.168.1.61:5060>
Call-ID: 1-15554@192.168.1.60
CSeq: 1 INVITE
Contact: <sip:sipp@192.168.1.60:5060;user=phone>
Max-Forwards: 70
Content-Type: multipart/mixed;boundary=boundary
Content-Length:  466
…
…
…
```

Example 15 Change and Remove Lines from SDP

In this sample of changing and removing lines from the SDP, the goal is to convert the G.729 codec to G.729a. The SIP manipulation would be configured like the following:

```
sip-manipulation
        name                            std2prop-codec-name
        description                     rule to translate standard to
proprietary codec name
        header-rule
                name                            CTypeManp
                header-name                     Content-Type
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        any
                new-value
                methods
                element-rule
                        name                            g729-annexb-no-std2prop
                        parameter-name                  application/sdp
                        type                            mime
                        action                          find-replace-all
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value                     a=rtpmap:[0-9]{1,3}
(G729/8000/1\r\na=fmtp:[0-9]{1,3} annexb=no)[[:1:]]
                        new-value                       G729a/8000/1
```

The result of manipulating the original SIP INVITE (shown above) with the configured SIP manipulation is:

```
INVITE sip:service@192.168.1.61:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.60:5060;branch=z9hG4bK-1-0
From: sipp <sip:sipp@192.168.1.60:5060>;tag=1
To: sut <sip:service@192.168.1.61:5060>
Call-ID: 1-15554@192.168.1.60
```

```
CSeq: 1 INVITE
Contact: <sip:sipp@192.168.1.60:5060;user=phone>
Max-Forwards: 70
Content-Type: multipart/mixed;boundary=boundary
Content-Length:  466
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=audio 12345 RTP/AVP 8
a=rtpmap:18 G729a/8000/1
a=sendrecv
a=maxptime:200
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=video 12345 RTP/AVP 34
a=rtpmap:34 H263a/90000
--boundary-
```

Example 16 Change and Add New Lines to the SDP

In this sample of changing and adding lines from the SDP, the goal is to convert non-standard codec H.263a to H.263. The SIP manipulation would be configured like the following:

```
sip-manipulation
        name                            prop2std-codec-name
        description                     rule to translate proprietary to
standard codec name
        header-rule
                name                            CodecManp
                header-name                     Content-Type
                action                          manipulate
                comparison-type                 case-sensitive
                match-value
                msg-type                        any
                new-value
                methods
                element-rule
                        name                            H263a-prop2std
                        parameter-name                  application/sdp
                        type                            mime
                        action                          find-replace-all
                        match-val-type                  any
                        comparison-type                 case-sensitive
                        match-value                     a=rtpmap:([0-9]{1,3})
H263a/.*\r\n
```

```
                          new-value                       a=rtpmap:+$1+"
H263/90000"+$CRLF+a=fmtp:+$1+" QCIF=4"+$CRLF
```

The result of manipulating the original SIP INVITE (shown above) with the configured SIP manipulation is:

```
INVITE sip:service@192.168.1.61:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.60:5060;branch=z9hG4bK-1-0
From: sipp <sip:sipp@192.168.1.60:5060>;tag=1
To: sut <sip:service@192.168.1.61:5060>
Call-ID: 1-15554@192.168.1.60
CSeq: 1 INVITE
Contact: <sip:sipp@192.168.1.60:5060;user=phone>
Max-Forwards: 70
Content-Type: multipart/mixed;boundary=boundary
Content-Length:  466
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=audio 12345 RTP/AVP 8
a=rtpmap:18 G729/8000/1
a=fmtp:18 annexb=no
a=sendrecv
a=maxptime:200
--boundary
Content-Type: application/sdp
v=0
o=user1 53655765 2353687637 IN IP4 192.168.1.60
s=-
c=IN IP4 192.168.1.60
t=0 0
m=video 12345 RTP/AVP 34
a=rtpmap:34 H263/90000
a=fmtp:34 QCIF=4
--boundary-
```