Reference Guide

Generic Header Manipulation
&
Regular Expressions

For the Ingate SIParators® and Firewalls using software release 4.10.x or later.

This document assumes a basic understanding of regular expressions and their behaviour.

1/22/2024

Revision History:

Revision	Date	Author	Comments
…	…	…	…
15	2020-01-20	PD	Complete re-write. “variable substitution” → “Header Access Variables”. Reduction → Normalization. CHM → CHO.
16	2020-04-26	PD	Tagged new 6.3 features
18	2020-10-16	PD	$(body.plain) and response (?!) corrections, HAV corrections.
19	2020-11-02	PD	Fix methods tag
20	2020-12-09	PD	Added $(count… call count variables plus minor fixes and adjustments
21	2021-11-12	PD	Clarify indices usage
22	2022-05-12	PD	Add ;b2buafwdref tag
23	2024-01-22	PD	Clarify .userinfo with RFC4904 tags

Table of Contents

Errata for Ingate <= 5.0.11 ..... **5**

1 Introduction **6**
1.1 Example 6
1.2 How? 6
1.3 Why GHM? 7
1.4 Where do I use GHM? 7
1.5 What can I do with GHM? 7

2 Where to configure GHM **8**
2.1 SIP Traffic – Dial Plan 9
2.2 Regexp match – Matching From Header 9
2.3 Regexp match – Matching R-URI 9
2.4 Regexp substitution and GHM – Forward To 9
2.5 SIP Traffic – Routing 10
2.6 SIP Trunk – Trunk 1-n 10
2.7 Incoming messages 11
2.8 Outgoing messages 12

3 Regular Expressions – matching your input **14**
3.1 Introduction 14
3.2 Explanation 15
3.3 Standard regular-expression notation 15
3.4 Escape special characters 16
3.5 Routing calls using the Dial Plan and the SIP trunk Page 16
3.6 Example Regular Expressions in the Matching From Header 17
3.7 Regular Expressions in the Matching Request URI 17
3.8 Regular Expressions in the Forward To 20
3.9 Examples of Basic Regular Expressions 20
3.10 Additional information 21

4 Special Tags in the Ingate Firmware **22**
4.1 Rewrite From header on egress 22
4.2 183 to 180 conversion 22
4.3 Do not REGISTER to trunk server(s) 22
4.4 Do not automatically monitor trunk server(s) with SIP OPTIONS 23
4.5 Force B2BUA on 23
4.6 B2BUA with media via the main dial plan 23
4.7 Explicitly state transport 23
4.8 Specifying Escape Characters (dial string) for e.g. Telia SIP trunk 24
4.9 Explicitly handle only specific METHODS 24
4.10 Force a specific response, e.g. 503, 5xx, 6xx 24
4.11 Support q-value in Trunk User Name 25
4.12 Support parallel forward in the dial-plan 26
4.13 Forward REFER through the B2BUA 26

5 Header Access Variables **28**
5.1 Headers 28
5.2 Body Access Variable 29
5.3 The difference between `$(x.user)` and `$([x.user])` 30
5.4 Port and Password 30
5.5 Indices, indexes, [?] 30

5.6 Example equivalences of a From header (built using HAV) 32
5.7 Examples 32

6 Call Count Variables 34
6.1 Call Counters 34
6.2 Call Count Logic 34

7 Conditionals 36
7.1 Conditional Regular Expressions (CRE) 36
7.2 Conditional Header Output (CHO) and Conditional Body Output (CBO) 41
7.3 Conditional Test (CT) 42
7.4 Conditional Results (CR) 45
7.5 Conditional Actions (CA) 46
7.6 Conditional Output (CO) 47
7.7 Conditional Header Output (CHO) examples 48
7.8 URI Parameter Chaining 51
7.9 Keyword / Grammar and Syntax Summary for CHO and CRE 52

8 Generic Header Manipulation (GHM) 53
8.1 GHM for Requests(?...&…) 54
8.2 GHM for Responses(?!...&!…) 55
8.3 GHM for Requests (?...&…) and Responses(?!...&!) combined in one expression 56
8.4 Multiple Occurrences of the same Header 56
8.5 Header Access Variables 57

9 Supplementary examples from real-world support cases 59

Terminology used in this document:

Abbreviation	Term
GHM	Generic Header Manipulation
CRE	Conditional Regular Expressions
CHO	Conditional Header Output
CHM	Conditional Header Manipulation
B2BUA	Back to Back User Agent
ITSP	Internet Telephony Service Provider
HAV	Header Access Variables

Text written in a `Monospace Font` generally signifies regular expression code i.e. GHM, CHO, CRE, HAV or their in/output, i.e. SIP URIs.

Note: Samples in this document are not guaranteed to be protocol-accurate or compliant; they are intended for demonstration only and are often reduced for simplicity.

Errata for Ingate <= 5.0.11:

You're using an *old* firmware – upgrade!

Note: In firmware versions <= 5.0.11 you cannot use `$1?From=$(...$1...)` – i.e. regexp capture result groups (on the trunk page i.e. where `$1` is before *and* after the "?"), *and* `$REGMATCH` expressions which also contain their own `$1` capture groups. The workaround is to use `$0?From=$(...$1...)`. The fix is to upgrade to >= 6.0.1.

Expressions such as the following will fail to evaluate:

```
sip:$1@192.168.1.1?From=%3Csip%3a$(REGMATCH_^001([0-9]{10})$_REGMOD
_+1$1_REGELSE_^1([0-9]{10})$_REGMOD_+1$1_REGELSE_([0-9]{10})$_REGMO
D_+1$1_REGEND.from.user)%40$(from.host)$([from.uriparams]) %3E$(fro
m.params)
```

A workaround is thus:

```
sip:$0?From=%3Csip%3a$(REGMATCH_^001([0-9]{10})$_REGMOD_+1$1_REGELS
E_^1([0-9]{10})$_REGMOD_+1$1_REGELSE_([0-9]{10})$_REGMOD_+1$1_REGEN
D.from.user)%40$(from.host)$([from.uriparams]) %3E$(from.params)
```

`$0` evaluates to the whole user portion of the RURI on the trunk page.

Or not to use a capture group on the trunk page e.g.

Incoming Trunk Match	Forward to
`(.*)`	`$1`

While instead use:

Incoming Trunk Match	Forward to
`.*`	`$0`

If a fix is necessary e.g. you must forward trunk captures to a host which differs from the PBX configured on the trunk page, write to support@ingate.com for a patch for your current firmware version <= 5.0.11, or upgrade your firmware > 5.0.11

1 Introduction

This document describes how to use Generic Header Manipulation (GHM) and Regular Expressions (RegEx) in an Ingate SIParator/Firewall. With the GHM feature it is possible to Add, Change or Remove any Header in SIP Requests and Responses.

1.1 Example

An ITSP requires the presence of a `P-Asserted-Identity` Header as part of authorization and the IP-PBX does not provide it. As seen here (ingress):

```
INVITE sip:6135551212@209.216.177.59:5060 SIP/2.0
Via: SIP/2.0/UDP 209.249.3.100:5060;branch=z9hG4bK9624349
To: sip:6135551212@209.249.3.56:5060
From: <sip:5035551111@209.249.3.100>;tag=3462103187-665679
Supported: timer, 100rel
Call-ID: 2165939-3462103187-665672@NXT02.broadvox.net
CSeq: 1 INVITE
Allow: INVITE, BYE, OPTIONS, CANCEL, ACK, REGISTER, PRACK, UPDATE
Max-Forwards: 69
Session-Expires: 3600;refresher=uac
Contact: sip:5035551111@209.249.3.100:5060
Content-Type: application/sdp
Content-Length: 249
```

The Ingate adds the header to the SIP message upon forward (egress) to the ITSP:

```
INVITE sip:6135551212@209.216.177.59:5060 SIP/2.0
Via: SIP/2.0/UDP 209.249.3.100:5060;branch=z9hG4bK9624349
To: sip:6135551212@209.249.3.56:5060
From: <sip:5035551111@209.249.3.100>;tag=3462103187-665679
Supported: timer, 100rel
Call-ID: 2165939-3462103187-665672@NXT02.broadvox.net
CSeq: 1 INVITE
Allow: INVITE, BYE, OPTIONS, CANCEL, ACK, REGISTER, PRACK, UPDATE
Max-Forwards: 69
Session-Expires: 3600;refresher=uac
Contact: sip:5035551111@209.249.3.100:5060
P-Asserted-Identity:sip:5035551111@64.156.174.74
Content-Type: application/sdp
Content-Length: 249
```

Note: The above is also performed on the trunk page by filling in the **Identity column** with `5035551111@64.156.174.74.`

1.2 How?

The following GHM expression can do the modification to make the above SIP request:

```
?P-Asserted-Identity=sip%3a$(from.user)%4064.156.174.74
```

Where `?` signifies a GHM – read more about these in the section Generic Header Manipulation (GHM). `P-Asserted-Identity` is the name of the header you wish to *Manipulate* – in this case,

create, or add to your egress SIP. `$(from.user)` is a variable. Read more about these in the section Header Access Variables.

1.3 Why GHM?

The purpose of GHM is to enhance the interoperability between different vendor equipment scenarios, as IP-PBX, Service Providers, and SIP enabled device OEMs implement SIP standards differently. GHM can normalize deviations.

With regular expressions, it is possible to match content from ingress SIP messages. When forwarding SIP messages, they can be rewritten to your specification. Header Access Variables make it possible to read information from a header at ingress in order to construct a new or replacement header at egress.

1.4 Where do I use GHM?

Rules are configured in the Dial Plan and SIP Trunk Page GUIs. The rules are configured in the same fields as regular expressions are written, i.e. telephone numbers, SIP from and to addresses. Read more about this in the section Where to configure GHM.

1.5 What can I do with GHM?

The GHM feature makes it possible to Add, Re-write or Delete any SIP Header of incoming or outgoing SIP messages, both Requests (Methods: `INVITE`, `ACK`, …) and Responses (`180 Ringing`, `200 OK`, …). GHM does not yet allow for write manipulation of individual parts of the ingress SIP header itself, but requires that a new header be created based on parts of SIP headers at ingress, and from arbitrary strings.

The following list summarizes available GHM actions:

- Add a header to an egress message, where a header doesn't exist at ingress
- Modify header content (by writing a new header)
- Replace an instance of a header
- Pass a header unchanged
- Delete a single or multiple instances of a header
- Delete multiple headers

Additionally, all listed actions can be done conditionally – i.e. check for a matching condition, and then act if the condition is met. Header manipulation for requests are performed after routing of calls, while responses are modified prior to routing.

2 Where to configure GHM

Regexp *matches* of SIP are configured in the below listed locations within the Ingate Firewall/SIParator, marked with blue, below.

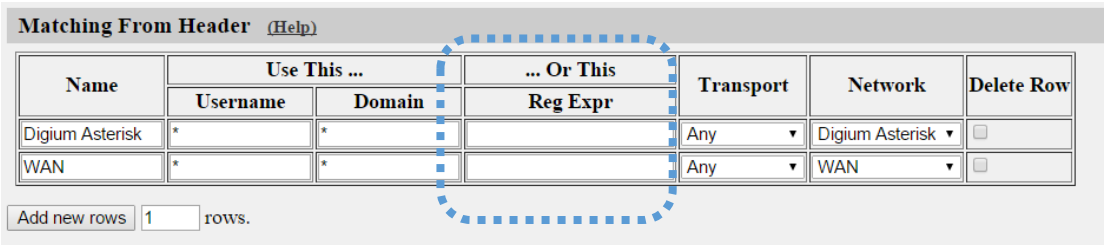
Regular Expression match
SIP Traffic > Dial Plan
Matching From Header – Reg Expr
Matching Request-URI – Reg Expr
SIP Trunks > Trunk n page
Main Trunk Line – Incoming Trunk Match
PBX Lines – From PBX Number/User and Incoming Trunk Match
SIP Lines – From SIP Number/User and Incoming Trunk Match

Regexp *substitution* and GHM of SIP are configured in destination fields, i.e. the *Forward To* fields in the below listed locations within the Ingate Firewall/SIParator, marked with red, below.

Regular Expression Substitution and GHM
SIP Traffic > Dial Plan
Forward To – Reg Expr
SIP Traffic > Routing
User Routing – Forward To
Static Registrations – Forward To
SIP Trunks > Trunk n page
Main Trunk Line – Username and Forward
PBX Lines – Username and Forward To PBX Account
SIP Lines – Username and Forward To SIP Account

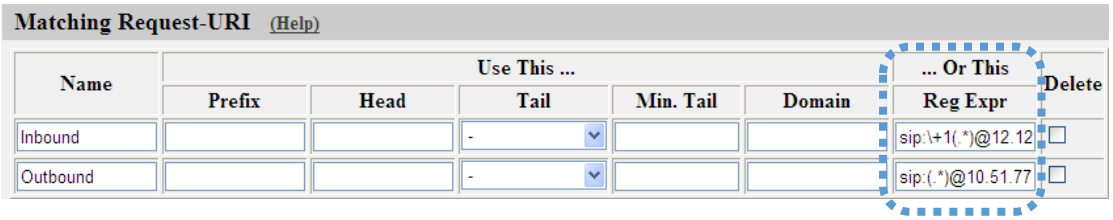
2.1 SIP Traffic – Dial Plan

2.2 Regexp match – Matching From Header



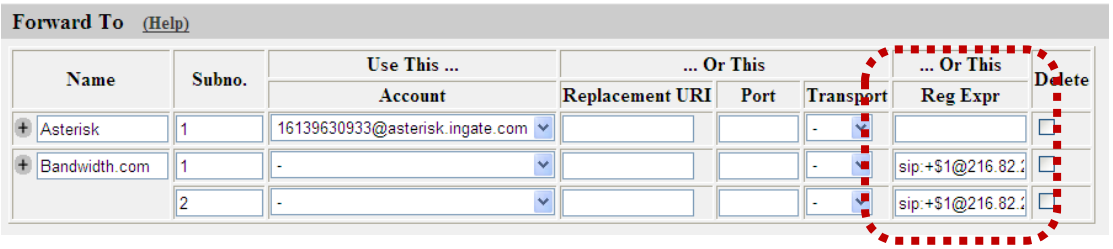
The *Matching From Header* matches source SIP URI, source Transport, and Network Address. In the Regular Expression, your expression defines matches for the From Header SIP URI of SIP messages. For a request to match, all criteria must be fulfilled.

2.3 Regexp match – Matching R-URI



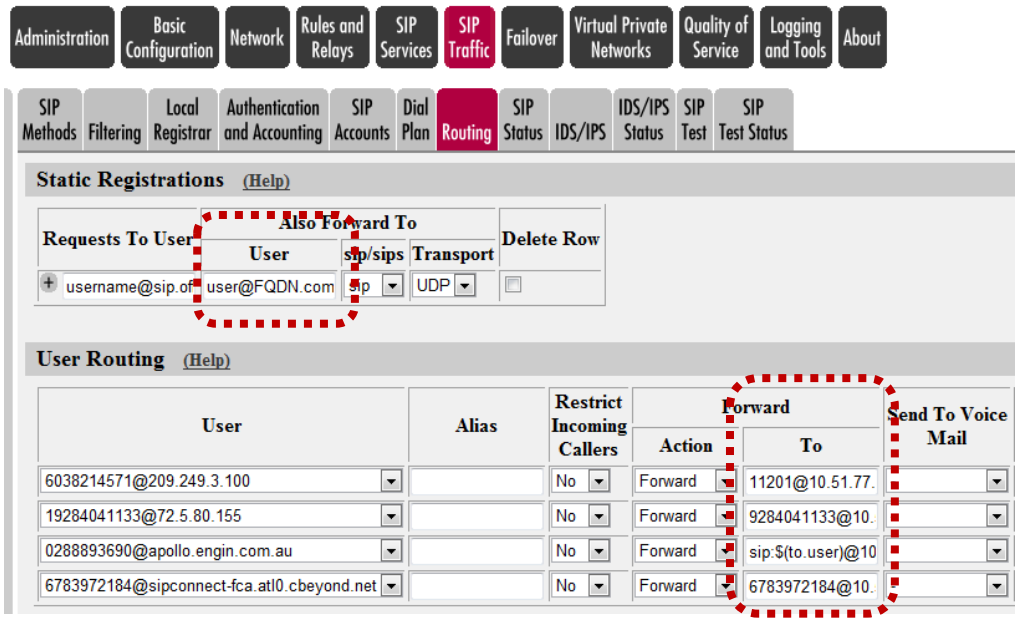
The *Matching Request URI* matches the incoming Request URI Header of ingress SIP messages. Typically, the "domain" portion of the URI is the Ingate IP Address or FQDN. Port and Transport can be used, but a match only occurs if both port and transport parameters are in the SIP RURI.

2.4 Regexp substitution and GHM – Forward To



The *Forward To* attribute of the Dial Plan defines where to send the SIP traffic. A specific destination SIP URI address is defined to forward the call to. Here you may enter Regular Expressions for the Dial Plan, used to define where the Ingate should forward the request using the Dial Plan. Here you define GHM and use HAV.

2.5 SIP Traffic – Routing

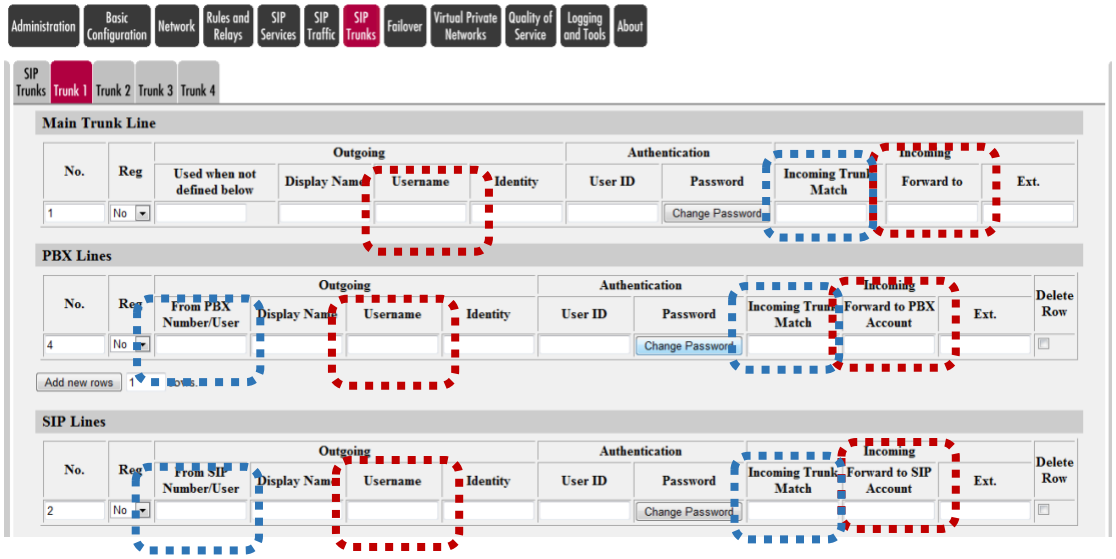


With Static Registrations, a certain user@address will additionally be redirected (forked) to another or more user@address. Even if an address is configured to be forwarded, the SIParator will contact the original addressee.

In the respective *Forward To* field you may use HAV and GHM, but not regular expressions.

2.6 SIP Trunk – Trunk 1-n

Remember: blue = match, red = substitute and GHM



From PBX/SIP Number/User - A regular expression – matches PBX extension, or user. For outgoing calls from the PBX to the ITSP, this field matches the From SIP URI. The row that matches first, is used for the outgoing call.

User Name - The SIP user name or phone number to use in the From SIP URI for outgoing calls and registrations towards the ITSP. This is often the telephone number of the ITSP SIP account and

usually the number displayed as caller ID on the PSTN. Here, you can use the result of a sub-expression from a match in a regular expression defined in the "From Number/User" field on the same row.

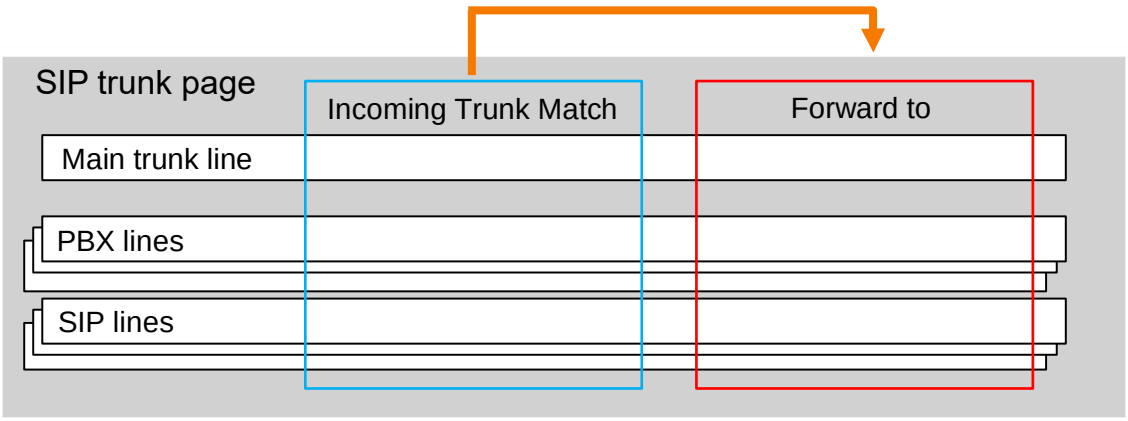
Identity – This field is for filling a `P-Asserted-Identity` or `P-Preferred-Identity` header in egress messages to your ITSP. You can choose on the trunk page which of the above two variants is sent.

Note: The GHM `$1?P-Asserted-Identity=__remove` removes `P-Asserted-Identity` headers from ingress messages upon egress.

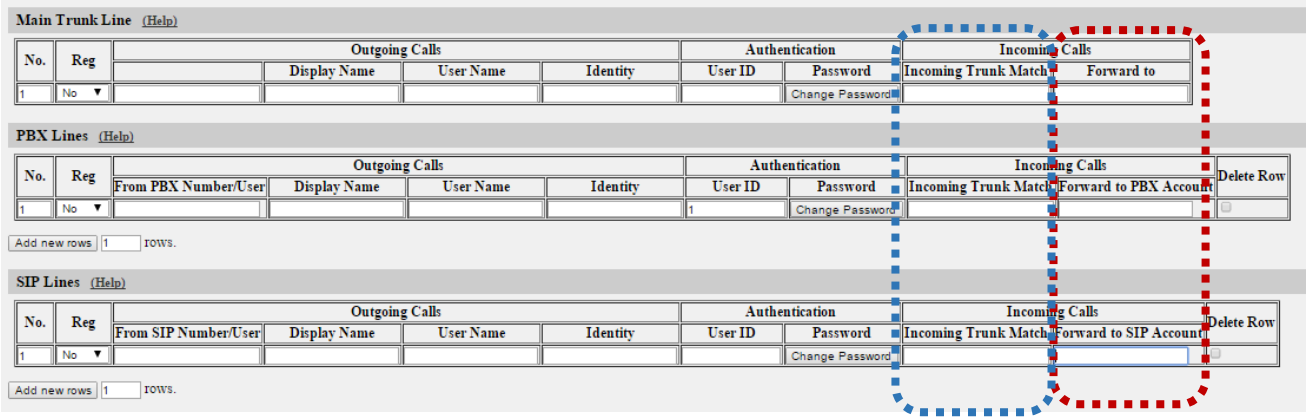
2.7 Incoming messages

The SIP Trunk page is designed to connect IP-PBXs and other SIP endpoints (phones) to an ITSP SIP Trunking service (using a B2BUA) with one interface to the ITSP and the other to the IP-PBX and SIP phones.

For incoming SIP messages, the rules for header manipulation of SIP messages are configured in the GUI for the SIP Trunk Page, in the columns of **Incoming Trunk Match** and **Forward to** for **PBX Lines**, **SIP Lines**, and **Main Trunk Line**.



For every incoming SIP INVITE, each row in the column **Incoming Trunk Match,** for **PBX Lines** and **SIP Lines**, is checked until match. No match? The values from **Main Trunk Line** are used.



When there is a match in **Incoming Trunk Match** field, the call will be forwarded according to the values in the respective **Forward to** field of the same row.

For **PBX Lines**, this is a user to which an incoming call from the ITSP will be forwarded to on the PBX.

For **SIP Lines**, this is an arbitrary SIP URI or a SIP user at a domain configured under Local Registrar, to which an incoming call will be forwarded.

Regular expressions can be used in the **Incoming Trunk Match** field to catch information from the R-URI to be used in the **Forward to** field, where also HAV and GHM can be made.
An example of a regular expression is:

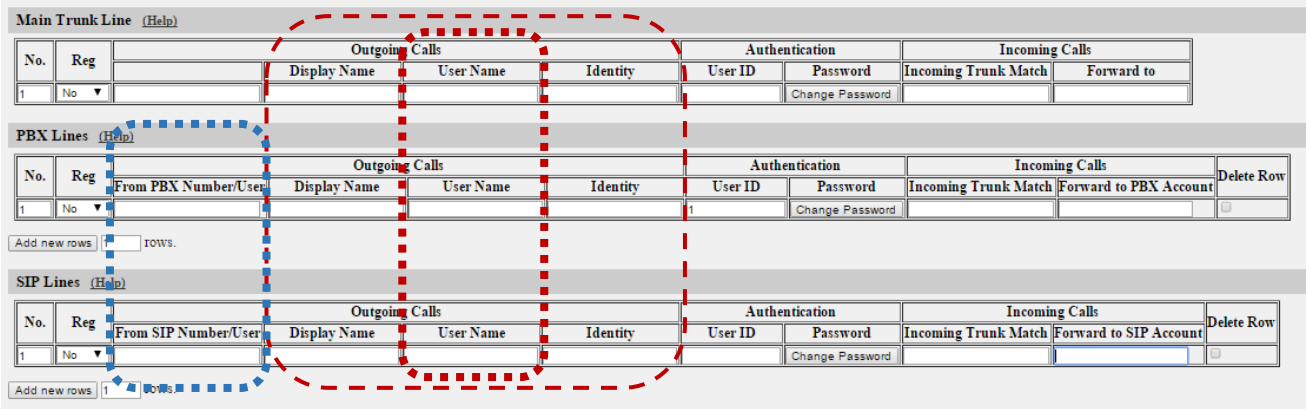
Incoming Trunk Match	Forward to
`\+1306(7707[0-9]{2})`	`0$1`
`\+1306(7707[0-9]{2})`	`0$1;user=phone`

This will result in an incoming call to +1306770713, forwards to the PBX number 0770713. Here you may also add URI parameters such as "`;user=phone`" at the end of the domain of a complete URI.

2.8 Outgoing messages

Outgoing calls are processed through the Dial Plan. The Dial Plan table is searched line by line from the top for a match from the PBX of the dialed number, where a SIP Trunk page is selected.

At the SIP Trunk Page, the caller's number (the user part of the From header) will - row by row, from top to bottom - be checked against the numbers or regular expressions entered in the column **From PBX/SIP Number/User**. A match will cause the trunk account under **User Name** to be used for the outgoing call.



The results of subexpressions from a match in the column **From PBX/SIP Number/User** can be used in the columns **Display Name, User Name** and **Identity**.

Display Name – This field modifies the "`display`" portion of your egress `From` header. E.g. entering `look at me` in the *Display Name* field gives:

```
From: "look at me" <sip:alpha@some.com>;tag=99
```

User Name – This field modifies the `user` portion of your egress `From` header. E.g. entering `no_wait_look_at_me` in the *User Name* field gives:

```
From: <sip:no_wait_look_at_me@some.com>;tag=64
```

Identity – This field adds a `P-Asserted-Identity` or `P-Preferred-Identity` header in egress messages to your ITSP. You choose on the trunk page which of the above two variants is sent. E.g. entering `bravo@real.com` in the *Identity* field gives:

```
From: "Display" <sip:alpha@fake.com>;tag=27
P-Asserted-Identity: bravo@real.com
```

Note: The GHM `$1?P-Asserted-Identity=__remove` when placed in any of the above three fields removes `P-Asserted-Identity` headers from ingress messages (e.g. from PBXes) upon egress.

To understand how GHM works it is first necessary to understand how regular expressions are used, see the chapter Regular Expressions – matching your input.

3 Regular Expressions – matching your input

Within the SIP Protocol (RFC 3261), a SIP URI identifies a communications resource. Like all URIs, SIP URIs may be placed in web pages, email messages, or printed literature. They contain sufficient information to initiate and maintain a communication session with the resource. In its simplest form a SIP URI looks like `sip:user@host`, where the `user` is the identifier of a particular client resource at the host being addressed. The term `host` in this context frequently refers to a domain, a network location.

Regular expressions increase the capabilities of the Ingate Dial Plan and SIP trunk. The Regular Expression is used to match the user, host and other parts of the SIP URI.

3.1 Introduction

Why regular expressions? To find our input! Then once we've matched (found) what we are looking for, do something with it. Perhaps you've encountered command-line tools such as `sed`, or `awk`. They all use regular expression notation. Let's say we want to find the text string:

```
small fluffy dog
```

Then once we've found it, modify it. We want to modify it to:

```
big fluffy dog
```

We don't want to do this manually every time – we want to automate this, and be sure that our automation will process only exactly those matches we are looking for – no more, no less. Now we know what we want to do – we just need to generalise, or specify, our **regular expression** which will look for matches. These **regular expressions** go in fields marked in BLUE in the section Where to configure GHM :

```
small (fluffy dog)
```

Let's say that we now have a match. Let's use a **regular expression substitution** – this is just a text string which expresses what we captured, at the earlier step, and includes any new, additional custom text – these **regular expression substitutions** go in fields marked in RED in the section Where to configure GHM:

```
big $1
```

This is the same process when dealing with SIP and telephone numbers. Identify the nature of the request (country code for least cost routing? is it an emergency number?):

```
sip:+46812345678@xyzcorp.com
```

Then once we've found it, we want to modify it to:

```
sip:0812345678@xyzcorp.com
```

We might do that with this regular expression:

```
sip:\+46([0-9]{9})@xyzcorp.com
```

Then get our intended result with the following **regular expression substitution**:

```
sip:0$1@xyzcorp.com
```

3.2 Explanation

`(.*)` Matches and stores any amount of characters in `$1`
`sip:(.*)@(.*)` Match and store `user` in `$1` and `host` in `$2` if applied to input string `sip:user@host`

Sub-expressions are ordinal to their starting parenthesis and referred to by $number. In other words, the *order* and *hierarchy* of the parentheses determines the *regular expression substitution order* E.g.

Expression	Matches String	Produces

`(((dog) pig) cat)`	`dog pig cat`	`$0 = dog pig cat`
		`$1 = dog pig cat`
		`$2 = dog pig`
		`$3 = dog`

In the expression `sip:(.*)@(.*)` which matches any Request-URI like `sip:user@ingate.com`, there are two referable sub-expressions: `user`, which is held in `$1`, and `ingate.com`, which is held in `$2`.

Sub-expressions can also be nested, as in the expression `(sip:(.*))@ingate.com`, which matches any Request-URI like `sip:user@ingate.com`, there are two referable sub-expressions: `sip:user`, which is referred to as `$1`, and `user`, which is referred to as `$2`.

3.3 Standard regular-expression notation

The Regular Expression flavor used in Ingate SIParator/Firewall is “POSIX Extended Regular Expressions (ERE)”.
Note: Character matches are case sensitive.

Operator	Description
`[]`	Matches any single character that is contained within the brackets. For example: `[abc]` Matches any single character in the set a, b, or c. `[a-z]` Matches any single character in the range a-z but not A-Z `[1-8]` Matches any single character in the range 1 to 8. `[369]` Matches any single character in the set 3, or 6, or 9.
`[^ ]`	Matches any single character that is not contained within the brackets. For example: `[^abc]` Matches any single character not in the set a, b, or c.
`.`	Matches any single character.
`,`	Matches the minimum specified characters or more.
`[0-9]`	Matches any decimal digit.
`[^0-9]`	Matches any non-digit.
`\s`	Matches any whitespace character.

\S	Matches any non-whitespace character.
\w	Matches any word (alphanumeric) character.
\W	Matches any non-word (alphanumeric) character.
(abc)	(abc) Matches the sequence abc and stores it as a variable which may be used in later expressions. (and) are also used for grouping.
$1	The $ symbol is used to recall expressions that have been stored via ()-variables which are numbered according to the capture hierarchy. $1 refers to the first variable stored, and $2 refers to the second variable stored, etc.
a|b	Matches a or b
+	Matches the preceding expression one or more times.
?	Makes preceding expression optional; if the preceding is inside () brackets, e.g. (345)? then that 345 is optional.
*	Matches the null string or any number of repetitions of the preceding expression.
{m}	Matches exactly m repetitions of the preceding expression.
{m,n}	Matches from m to n (inclusive) repetitions of the preceding expression.
{m,}	Matches m or more repetitions of the preceding expression.
^	Matches the start of the string.
$	Matches the end of the string.

3.4 Escape special characters

Meta characters are characters with a special meaning in Regular Expressions. There are a number of characters with special meanings: \ ^ $. | ? * + ( ) [ ] { } . If you want to use any of these characters as a literal in a Regular Expression (i.e. to find these exact characters in a source string), you need to *escape* them in your RegExp by using a backslash. For example, + is escaped as \+ and so to match +46701234567 we use \+46701234567.

Expression	Does Not Match String	Produces
sip:+46(5552345)@corp.com	sip:+465552345@corp.com	$1 =

Expression	Matches String	Produces
sip:\+46(5552345)@corp.com	sip:+465550505@corp.com	$1 = 5550505

Why? In the expression sip:+46 the portion + is a quantifier for the colon character, so :+ looks for 1 (one) colon character.

3.5 Routing calls using the Dial Plan and the SIP trunk Page

The dial plan and the SIP Trunk pages dictate how to route calls. From whom to accept calls, and to where to send calls. Using regular expressions in the dial plan and the SIP trunk page allows you to generically specify a range of numbers, range of domains, or other set of specific digits.

Regular expressions are a flexible way of delivering patterns that match a unique set of criteria. For example, if you specify the regular expression [0-9]{7,} Ingate Firewall/SIParator recognizes seven or more instances of digits zero to nine. In other words, a telephone number.

3.6 Example Regular Expressions in the Matching From Header

The purpose of the *Matching From Header* table is to narrow the source selection at ingress. It exclusively examines the `From` header.

SIP URI Example Description	Equivalent Regular Expression

7-digit number @ Any Domain or IP
7-digit number @ IP Address
7-digit number @ Domain
	`sip:[0-9]{7}@.*`
	`sip:[0-9]{7}@12.34.56.78`
	`sip:[0-9]{7}@sip_domain.com`

North American Toll-free number: 1+800, 1+866, 1+877, 1+888+7 digits @	`sip:18(00|66|77|88)[0-9]{7}@`
7-digit number, beginning with optional 9 @	`sip:9?[0-9]{7}@`
4-digit number (as an extension) starting with 5 @	`sip:5[0-9]{3}@`
4-digit number not starting with 36 @	`sip:(?36)[0-9]{4}@`

Anyone @ Anywhere
Anyone @ IP Address
Anyone @ Domain
	`sip:.*@.*`
	`sip:.*@12.34.56.78`
	`sip:.*@sip_domain.com`

7-digit numbers within the London area codes 0207 and 0208@	`sip:020[78][0-9]{7}@`

7-digit number with 0845 prefix @ 6 to 7-digit number with 0845 prefix @
6-digit number with 0845 prefix @
	`sip:0845[0-9]{7}@`
	`sip:0845[0-9]{6,7}@`
	`sip:0845[0-9]{6}@`

7-digit number with 0845 or 0870 prefix @ **Note**: the first will also match 0875 or 0840, the second won't	`sip:08[74][05][0-9]{7}@` `sip:(0870|0845)[0-9]{7}@`
Any 9 to 10-digit numbers prefixed with optional 00 and then mandatory 44	`sip:0?0?44[0-9]{9,10}@`

From Ingate >= 6.2.0, RegEx matches captured via the *Matching From Header* table can be later accessed in the *Forward To* RegExp field via `$fx`, i.e. `$f1`, `$f2` etc where f signifies From. For example:

Expression	Matches String	Produces
`sip:(5552345)@corp.com`	`sip:5552345@corp.com`	`$f1 = 5552345`
`sip:(555([0-9]{4}))@corp.com`	`sip:5550505@corp.com`	`$f1 = 5550505` `$f2 = 0505`

3.7 Regular Expressions in the Matching Request URI

The purpose of the *Matching Request-URI* table is to match a Request URI of the SIP messages at ingress to determine where it wants to go. To effectively determine routing. It exclusively examines the Request URI of ingress SIP requests (`INVITE`, `REGISTER`, …).

Typically, the "domain" portion of the URI is the Ingate IP Address or FQDN. Port and Transport can be specified, but the RegExp will only produce matches if both port and transport parameters exist in a URI at ingress.

SIP URI Example Description	Equivalent Regular Expression

7-digit number @ Any Domain
7-digit number @ IP Address
7-digit number @ Domain
	`sip:([0-9]{7})@.*`
	`sip:([0-9]{7})@12.34.56.78`
	`sip:([0-9]{7})@sip_domain.com`

Emergency numbers 112 or 999 @ ***Warning**: not all SIP providers have access to emergency service numbers.*	`sip:112|999@`
North American Toll-free number: 1+800, 1+866, 1+877, 1+888+7 digits @	`sip:18(00|66|77|88)[0-9]{7}@`
7-digit number, beginning with optional 9	`sip:9?[0-9]{7}@`
4-digit number starting with 5	`sip:5[0-9]{3}@`
4-digit number not starting with 36 @	`sip:(?36)[0-9]{4}@`
Remove Prefix "1613" on any Username @ Anything *$1 is provided by (.*) i.e. 1613 is matched but not stored. If the number doesn't begin "1613", there will be no match.*	`sip:1613(.*)@.*`
Remove optional Prefix "+" on any Username @ ***Note**: the "+" character is escaped to match*	`sip:\+?(.*)@`

Any Username @ Any Domain with Port and Transport – case sensitive
Any Username @ IP Address with Port and Transport – case sensitive
Any Username @ Domain with Port and Transport – case sensitive
	`sip:(.*)@.*:5060;transport=UDP`
	`sip:(.*)@12.34.56.78:5060;transport=UDP`
	`sip:(.*)@sip_domain.com:5060;transport=UDP`

7-digits within area codes 0207 and 0208@	`sip:020[78][0-9]{7}@`
7-digit number with 0845 prefix @ 6 to 7-digit number with 0845 prefix @	`sip:0845[0-9]{7}@` `sip:0845[0-9]{6,7}@`
6-digit or longer number with 0870 prefix @	`sip:0870[0-9]{6,}@`
7-digit number with 0845 or 0870 prefix @ **Note**: the first can also match 0875 or 0840, the second won't	`sip:08[74][05][0-9]{7}@` `sip:(0870|0845)[0-9]{7}@`
Any 9 to 10-digit numbers prefixed with optional 00 and then mandatory 44	`sip:0?0?44[0-9]{9,10}@`

Optional 353 prefix with or without optional 00 start, then optional 0 with optional 1-2 digit area code, then mandatory 7-digit number @ any domain. **Note:** The following URIs will match: sip:00353123456678@abc.com sip:35312345678@asdf.com sip:12345678@asdf.com sip:2345678@abc.com sip:012345678@domain.com sip:0212345678@wherever	sip:(((00)?353)?0?[0-9]{1,2})?([0-9]{7})@.*

Note: in all of the above expressions, there is no match if the RURI isn't pre-pended with `sip:`, i.e. requests beginning with `tel:` will not match. Also, while `sip:` is matched, it isn't stored in any of the above examples. Any SIP RURI at ingress which is not prefixed with `sip:` is not a valid SIP URI.

RegEx matches captured via the *Matching Request-URI* table can be later accessed in the *Forward To* RegExp field via `$x`, i.e. `$1`, `$2`.

3.7.1 Examples for a trunk

Let's say we own the trunk series 5550140 – 5550159, i.e. a range of twenty different extensions, 40-59. We want an expression which will match the range of 20 extensions, but only 20 extensions, and not 00-39 or 60-99.

Expression	Matches Strings	Produces
sip:(55501[0-9][0-9])@xy.com	sip:5550100@xy.com sip:5550163@xy.com sip:5550199@xy.com	$1 = 5550100 $1 = 5550163 $1 = 5550199
sip:(55501([4-5][0-9]))@xy.com	sip:5550140@xy.com	$1 = 555140 $2 = 40
sip:(55501([4-5][0-9]))@xy.com	sip:5550159@xy.com	$1 = 555159 $2 = 59

As a result, the appropriate regular expression in this worked case is either of the last two expressions.

3.7.2 Special Expressions for captures made in Request-URI

From Ingate >= 6.2.0, RegEx matches captured via the *Matching Request-URI* table can also be later accessed in the *Forward To* RegExp field via `$rx`, i.e. `$r1`, `$r2` etc where r signifies R-URI. For example:

Expression	Matches String	Produces
sip:(5552345)@corp.com	sip:5552345@corp.com	$r1 = 5552345

sip:(555([0-9]{4}))@corp.com	sip:5550717@corp.com	$r1 = 5550717 $r2 = 0717

3.8 Regular Expressions in the Forward To

A *Regular Expression Substitution* is used in the *Forward To* field. It refers to RegExp sub-expressions *matched and captured* in the *Matching Request-URI* table. Sub-expressions are numbered in the order of their starting parenthesis and referred to in `$number` fashion.

The *Forward To* attribute of the Dial Plan defines where and how to send SIP traffic. An arbitrary string, a RegExp Substitution, or combination thereof, is used to define a destination SIP URI.

You may define lines in the Dial Plan that lack a Forward to definition. This is useful if you for example are forwarding by ENUM.

SIP URI Example Description	Equivalent Regular Expression
Fixed number: 911 @ IP Address Fixed number: 911 @ Domain	sip:911@12.34.56.78 sip:911@sip_domain.com
Fixed 7-digit number @ IP Address	sip:9630933@12.34.56.78
North American long-distance number @ Domain	sip:16139630933@sip_domain.com
North American Toll-free number: 1+800+7 digits @ Domain	sip:18668090002@sip_domain.com
Use Stored Variable $1 @ IP Address Use Stored Variable $1 @ Domain	sip:$1@12.34.56.78 sip:$1@sip_domain.com
Use Stored Variable $1 @ Domain with Port and Transport	sip:$1@sip_domain.com:5060;transport=UDP
Add To Header from ingress message into Request URI @ Domain	sip:$(to.user)@sip_domain.com
Add To Header from ingress message into Request URI and To Host from ingress message into Domain and send to specified address	sip:$(to.user)@$(to.host)
Add +1 in front of To Header from ingress message in Request URI @ Domain	sip:+1$(to.user)@sip_domain.com

Note: The table includes examples of HAVs, e.g. `$(to.user)`, see the chapter Header Access Variables for explanation.

3.9 Examples of Basic Regular Expressions

Here are some basic examples of some standard Regular Expressions to be used in the *Forward To* columns either of the Dial Plan or the SIP Trunk page (without the use of the Generic Header Manipulation). The examples assume that there is a match done where `$1` contains the user part of the Request-URI from ingress.

Forward $1 to domain or IP	… specify destination port with :port
`sip:$1@192.168.1.1`	`sip:$1@192.168.1.1:5060`
`sip:$1@example.com`	`sip:$1@example.com:5060`
Force B2BUA on	… specify UDP transport via transport parameter
`sip:$1@192.168.1.1;b2bua`	`sip:$1@192.168.1.1;transport=UDP`
`sip:$1@example.com;b2bua`	`sip:$1@example.com;transport=UDP`
Add + Prefix	
`sip:+$1@192.168.1.1`	
`sip:+$1@example.com`	
Add Any Combination of the above	
`sip:$1@192.168.1.1:5060;transport=TCP;b2bua`	
`sip:$1@example.com:5060;transport=TCP;b2bua`	

3.10 Additional information

Here are a few resources we recommend to read more about to build and test regular expressions before they go live:

- http://www.regular-expressions.info/
- http://gskinner.com/RegExr/ or https://regexr.com/
- http://renschler.net/RegexBuilder/
- echo (Test expression) | grep -E (regexp)

4 Special Tags in the Ingate Firmware

4.1 Rewrite From header on egress

You can use the legacy method to re-write the `From` header on egress instead of the more complex GHM. Add `;from=` and a quoted, valid SIP URI to the end of your RegExp.

Forward To …or this… Reg Expr:

```
sip:$1@192.168.1.1;from="sip:+133355500000@1.2.3.4"
```

Note: the `From` header is the only header that can be changed in this legacy way i.e. using the `;uriparams` format. This way, the B2BUA is not engaged.

All headers can be added or changed via GHM, i.e. `?From=…`
See the chapter Generic Header Manipulation (GHM).

Do not use legacy `;from=` together with newer GHM `From=…` you will get unpredictable results.

4.2 183 to 180 conversion

If you want to convert a **183 Session Progress** at ingress from the ITSP to a **180 Ringing** on egress towards the PBX add `;cnv183` to the Domain Name or IP Address in the **Service Provider Domain** field at the SIP Trunk page. Note: this method should remove SDPs.

Service Provider Domain:

```
10.20.30.40;cnv183
```

4.3 Do not REGISTER to trunk server(s)

When using registration (REG = Yes) and a fallback domain, the Ingate will normally SIP REGISTER to both. Adding the `;no-reg` flag to either of the domains will skip registering to that domain.

Add `;no-reg` to the Domain Name or IP Address in the **Service Provider Domain** field at the SIP Trunk page. Requires firmware >= 6.0.3.

Service Provider Domain:

```
10.20.30.40;no-reg,10.20.30.41
```

This disables sending of REGISTER to a proxy when multiple proxies are entered in the Service Provider Domain.

4.4 Do not automatically monitor trunk server(s) with SIP OPTIONS

When using registration & a fallback domain, adding the `;no-mon` flag to either of the domains will skip auto monitoring that domain. Auto-monitoring commences after the Blacklisting timeout duration after the first successful registration to the ITSP.

To cease monitoring of an ITSP proxy domain, add `;no-mon` to the Domain Name or IP Address in the **Service Provider Domain** field at the SIP Trunk page. Requires firmware >= 6.0.3. In firmware >= 6.0.3, SIP Trunk servers are automatically monitored.

Service Provider Domain:

```
sip.itsp.com,sip2.itsp.com;no-mon
```

This disables sending of SIP OPTIONS to a proxy when multiple proxies are entered in the Service Provider Domain.

4.5 Force B2BUA on

Forward To …or this… Reg Expr:

```
$1@192.168.1.1;b2bua
```

4.6 B2BUA with media via the main dial plan

In order to relay media i.e. to anchor media at the SBC when e.g. releasing or diverting calls back to the operator – this option is synonymous with the "Relay media" trunk page option: on the main dial plan, add `;b2buawm`. A bug in later 5.0.x firmware series prevented this expression from working properly and was fixed in the 6.0.2 firmware. Note, you must have a matching RegExp for the `$1` parameter to be filled. This RegExp is used on the main Dial Plan, Forward To rows:

Forward To …or this… Reg Expr:

```
$1@10.20.30.40;b2buawm
```

4.7 Explicitly state transport

SIP RFC 3261 specifically states that two URIs are not synonymous if a port or implied parameter is absent in one but present in a second while all other parameters are equal. For implied parameters, such as transport, their presence – or absence – can be important. A UAC must specify its transport when registering to a proxy, if it is not UDP:

Reg Expr:

```
john@10.20.30.40;transport=tcp
```

Note that RFC3261 deprecates the use of transport=tls although its use on an Ingate is accepted for compatibility. If an ITSP `Contact:` header contains the transport parameter `transport=tcp` in one response, but removes it in another response – implying the SIP default `transport=udp` – the

transport will be deemed to have **changed** on the ITSP side, and the Ingate will contact the new destination. If the ITSP cannot handle this transport change (where two independent daemons listen for traffic via the two different transports which may be unaware of each other), this is an error that the ITSP must correct.

4.8 Specifying Escape Characters (dial string) for e.g. Telia SIP trunk

Reg Expr:

```
;escape-chars=*#
```

4.9 Explicitly handle only specific METHODS

Requires firmware >= 4.10.1

Reg Expr – as a tag to a regular expression:

```
;methods="ACK,INVITE,CANCEL,OPTIONS"
```

The above regular expression has the effect that **only** the named SIP methods `ACK`, `INVITE`, `CANCEL` and `OPTIONS` are to be handled. Note that BYE cannot be handled in the dial plan.

E.g. (under main dial plan)

Matching RURI:

```
sip:172.18.137.113@{0}
```

Forward To Reg Expr:

```
10.3.1.1;methods="OPTIONS"
```

The above expression has the effect that **only** the SIP method `OPTIONS` is handled/affected.

e.g. (under SIP Traffic → Routing → User Routing)

User:

```
bob@proxy.lan
```

Action:

```
Parallel
```

Forward to:

```
sip:bob@presence-server.corp.lan;methods="PUBLISH,SUBSCRIBE,INFO"
```

4.10 Force a specific response, e.g. 503, 5xx, 6xx

Requires firmware >= 6.2.2

Add `;respond="xxx"` to *Reg Expr* in *Forward To*. If found, a SIP response with the specified status code `xxx` is sent. Note: the code must be surrounded by quotes (`"xxx"`). It can take 1 parameter: `?Retry-After=yyy` which adds such a header to the response being sent.

Note: `?Retry-After` is not a real GHM, it just uses the same syntax.

If you wish to put the Ingate into a maintenance mode and reject new requests with 503 which include a `Retry-After` header with a value of 1800 seconds (30 minutes).

Forward To Reg Expr:

```
;respond="503"?Retry-After=1800
```

Add a new dial plan row at row 1 which uses only this *Forward To* and this will respond to all traffic with your specific code. For example:

Forward To

Name	No.	Use This ...	... Or This			... Or This	... Or This	Use Alias IP
		Account	Replacement Domain	Port	Transport	Reg Expr	Trunk	
SERVICE	1	-			-	`;respond="503"?Retry-After=1800`	-	-

Dial Plan

No.	From Header	Request-URI	Action	Forward To	Add Prefix		ENUM Root	Time Class	Comment
					Forward	ENUM			
1	-	-	Forward	SERVICE			-	-	SERVICE 503

If you wish to reject new requests with a 600 e.g.

Forward To Reg Expr:

```
;respond="600"
```

4.11 Support q-value in Trunk User Name

Requires firmware >= 6.2.0

Add the tag ;q=0.5 in the *User Name field* on the Trunk page to add the q-value as a parameter to the Contact header in REGISTERs sent from the Trunk page.

e.g.

Main Trunk Line

No.	Reg	Outgoing Calls			Authentication		Incoming Calls	
		Display Name	User Name	Identity	User ID	Password	Incoming Trunk Match	Forward to
1	No		`mytrunk;q=0.7`					

4.12 Support parallel forward in the dial-plan

Requires firmware >= 6.3.0

Add the tag `;parallel` in the "Reg Expr" field in the "Forward To" table in the "Dial Plan". This makes the Forward To sub-rows send in parallel (as opposed to send in sequence).

e.g.

Forward To

1 `;parallel` Trunk 1

2 `;parallel` Trunk 1

Or

Forward To

1 `sip:$1@192.168.1.10;parallel`

2 `sip:$1@192.168.1.11;parallel`

4.13 Forward REFER through the B2BUA

Requires firmware >= 6.4.1

Add the tag `;b2buafwdref` in the "Reg Expr" field in the "Forward To" table in the "Dial Plan". This works like `;b2bua` but forwards REFER through the B2BUA.

e.g.

Forward To

1	`;b2buafwdref`	Trunk 1
2	`;b2buafwdref`	Trunk 1

Or

Forward To

1	`sip:$1@192.168.1.10;b2buafwdref`
2	`sip:$1@192.168.1.11;b2buafwdref`

-End chapter-

5 Header Access Variables

5.1 Headers

The Ingate firmware has a range of ***variables*** for use in RegExp and GHM which provide (currently) read-only ***access*** to any ***header***, URI, or part thereof. Variables must be enclosed in `$(…)` or `$([…])`. Where a SIP URI is:
`dname <sip:user;tgparams:password@host:port;uriparams>;params`

Variable [URI.portion]		Explanation
`cfg`	`.user`	The user part of the Local Registrar account
	`.host`	As above, host part.
`ruri`	`.user`	User part of Request-URI
	`.host`	Host part.
	`.uriparams`	URI parameters.
`header_name`	`.user`	User part of `header_name` between `sip(s):` and `@`
These parts only work on headers which contain valid <sip:...> URIs e.g.: From, To, Request-URI, Route, Record-Route, Contact, etc.	`.userinfo`[3]	User, password, trunk-params, found at ingress up to the @ symbol.* Everything between `sip(s):` and `@`
	`.user`	User (i.e. phone number) in parent `.userinfo`
	`.params`	Any RFC4904 trunk-params in parent `.userinfo`
	`.password`	Any password in parent `.userinfo`
	`.cpassword`	Any colon prepended password in parent `.userinfo`
	`.userinfoat`[2]	As above*, but includes the final `@`.
	`.password`	Password part of "header_name" (e.g. `to.password`). *
	`.cpassword`[2]	As `password`; but output is prepended with colon (“:`password`”)[2]. Evaluates if `header_name .userinfo` contains a `password`. *
	`.host`	Host part of `header_name`
	`.port`	Port part of `header_name` **
	`.cport`[2]	As port, but prepended with a colon “:”. Evaluates if `header_name` contains a `port`. **
	`.dname`	Display name of `header_name` if present.
	`.dnameuri`[1]	Display name and URI of `header_name`.***
	`.uqdname`	Unquoted display name
	`.params`	Header parameters (outside of <sip:URI>) e.g. `;tag=asdf`
	`.uriparams`	Parameters within the URI: e.g. `;transport=udp`
	`.telnum`	tel: URI
	`.uri`	Whole URI string (after *`header_name:`*) of a header considered a URI ***
`hdr`	`.header_name`	The 1st ingress `header_name` instance content
	`.header_name[2]`	The 2nd ingress `header_name` instance content
`rawhdr`	`.header_name`	The unescaped ingress `header_name` content
`ip`	`.ethx`	The IP address of *eth* network interface, (e.g. ip.eth1)

[1] *(Ingate >= 5.0.4)* [2] *(Ingate > 6.0.0)* * Only supplies *user* if no password at ingress. **: returns null if no port value at ingress ***: includes <sip:...> if present at ingress, but lacks params [3] Sub-variables available. See below for example usage.

This is a SIP URI:

```
dname <sip:user:password@host;uriparams>;params
```

It might be constructed using these variables:

```
$(x.dname)
<sip:$(x.user):$(x.password)@$(x.host)$(x.uriparams)>$(x.params)
```

Alternatively:

```
$([x.dname])
<sip:$([x.user]):$([x.password])@$([x.host])$([x.uriparams])>$([x.p
arams])
```

Which would be formed by the encoded string:

```
$([x.dname])%20%3Csip:$([x.user]):$([x.password])%40$([x.host])$([x
.uriparams])%3E$([x.params])
```

This is a Header (not a URI):

```
Some-Header: Blah blah blah – something, version 1.2
```

This is a SIP URI with RFC4904 trunk group parameters:

```
dname <sip:user;tgrp=001;trunk-
context=example.com:password@host;uriparams>;params
```

It is constructed using these variables and sub-variables[3]:

```
$(x.dname)
<sip:$(x.userinfo.user)$(x.userinfo.params):$(x.userinfo.password)@
$(x.host)$(x.uriparams)>$(x.params)
```

5.2 Body Access Variable

Variable [URI.portion]	Explanation
body .plain[1]	Returns the whole body – this is normally the SDP. For multipart messages, this returns all content after the double CR/LF e.g.: `$([body.plain])` This is only intended to be useful to analyze the body.
[1] *(Ingate >= 6.2.0)*	

Example:

```
sip:$(REGMATCH_m=video_REGMOD_videouser@mycorp.com_REGELSE_.*_REGMO
D_audiouser@mycorp.com_REGEND.body.plain)
```

produces `videouser@mycorp.com` if the message body contains '`m=video`', otherwise it produces `audiouser@mycorp.com`

5.3 The difference between `$(x.user)` and `$([x.user])`

All HAV and BAV are invoked by enclosing the variable in $parentheses – i.e. the `$(variable)` syntax. The same is achieved by using `$([variable])` – the difference being that, if the returned content of `$([variable])` is empty, then the result is a zero-length string, i.e. no output. If `$(variable)` evaluates as empty, because for example its match did not exist at ingress, the literal string `$(variable)` is output. The `$([variable])` syntax is available from firmware >= 4.10.1. As a convenience, in firmware >= 6.2, an empty `$(variable)` returns a zero-length string, i.e. no output.

Note: To use any variables, a regular expression match must have been done (Main Dial Plan, Matching R-URI), together with a regular expression substitution. E.g. `$1?header=…`

If you only want to re-write the `From` header, use the legacy method in the *Forward To Reg Expr field* and append a "`;from=`" parameter.

5.4 Port and Password

Parameters `password` and `port` are properties of an address and shall be prepended by a colon. But a colon does not independently appear in a SIP URI without it causing a parsing problem. If you write the expression …`$(from.user)%40(from.host)%3a(from.port)`… and the ingress `From` header contained no port parameter, your expression evaluates to `123@192.168.1.1:` which is an invalid URI, since there is no port value after the colon. If it is unknown in advance whether port or password would be found in an ingress SIP URI, the following workaround CHOs (which require firmware >= 5.0.4) can help:

```
$(CONDIF.from.port)$(CONDYES.PLAIN.%3A)$(CONDYES.from.port)
or
$(CONDIF.from.password)$(CONDYES.PLAIN.%3A)$(CONDYES.from.password)
```

Convenience variables for the above are available in Ingate firmware >= v6.0:

`$([header_name.cport])`	yields e.g.	:5060
`$([header_name.cpassword])`	yields e.g.	:p@$$word

5.5 Indices, indexes, [?]

- index [0] is synonymous with: no index [] and index [1]; it gets the first instance of that header
- index [1] is synonymous with the first instance of that header (the one at the top)

- index [2] is synonymous with the second instance of that header
- index [-1] is synonymous with the last instance of that header (at the bottom)
- index [-2] is synonymous with the second to the last instance of that header.

e.g. Route headers rewritten as HAV:

```
Route: <sip:route[1].user@route[1].host;route[1].uriparams>
Route: <sip:route[2].user@route[2].host;route[2].uriparams>
Route: <sip:route[3].user@route[3].host;route[3].uriparams>
```

is the same as

```
Route: <sip:route.user@route.host;route.uriparams>
Route: <sip:route[2].user@route[2].host;route[2].uriparams>
Route: <sip:route[-1].user@route[-1].host;route[-1].uriparams>
```

is the same as

```
Route: <sip:route[-3].user@route[-3].host;route[-3].uriparams>
Route: <sip:route[-2].user@route[-2].host;route[-2].uriparams>
Route: <sip:route[-1].user@route[-1].host;route[-1].uriparams>
```

5.5.1 Limit the scope of operation

`?History-Info=__remove` removes all `History-Info` headers
`?History-Info[2]=__remove` removes only the second `History-Info` header

`?History_Info=something` sets all `History-Info` headers to the same value `something`
`?History_Info[2]=something` sets only the 2nd one to `something`, the 1st and 3rd remain unchanged

`?History_Info=__remove&History_info=something` removes all `History-Info` headers and only one new `History-Info` header with value `something` is created.

5.5.2 Example INVITE - URIs and headers rewritten as variable (HAV) names

```
INVITE sip:ruri.user@ruri.host:ruri.port;ruri.uriparams SIP/2.0
Via: via[1].uri
Via: hdr[2].via
To: "to.uqdname" <sip:to.user@to.host>
From: from.dname <sip:from.user@from.host>;from.params
Call-ID: call-id.user@call-id.host
P-Asserted-Identity: sip:P-Identity.user@P-Asserted-Identity.host
Contact: <sip:Contact.user@Contact.host>
Record-Route:
<sip:Record-Route[1].user@Record-Route[1].host;record-route.uripara
ms>
```

```
Record-Route:
<sip:Record-Route[-1].user@Record-Route[-1].host;record-route.uripa
rams>
Session-Expires: hdr.session-expires
User-Agent: hdr.user-agent
Supported: hdr.supported
Allow: hdr.allow
Max-Forwards: hdr.max-forwards
CSeq: 1 INVITE
...
```

5.6 Example equivalences of a From header (built using HAV)

```
From: from.dname <sip:from.user@from.host>;from.params
```

is the same as

```
From: "from.uqdname" <sip:from.user@from.host>;from.params
```

is the same as

```
From: from.dname from.uri;from.params
```

is the same as

```
From: hdr.from
```

5.7 Examples

Adding From Header
`sip:$1@192.168.1.1;from="sip:$(from.user)@1.2.3.4"`
`sip:$1@example.com;from="sip:$(from.user)@1.2.3.4"`
`;from="sip:$(from.user)@example.com"` **Replaces From domain at ingress with "example.com".*
Compare: `;from="sip:$f1@example.com"` *Replaces From domain at ingress with "example.com".* `$f1` *refers to the* Matching From *Reg Expr "(.*)@.*". While* `$f1` *can be part of the user string,* `$(from.user)` *is the whole user string.*

The above examples assume that there is a RegExp match and capture done at an earlier stage – the RegExp fills `$1` with a result – this result is a substitution – here, the user part of the Request-URI.

Note: the `From` header is the only header that can be added or changed in the legacy fashion, e.g. using the `;from=` format. All headers can be added through GHM, including the `From` header. See the chapter Generic Header Manipulation (GHM).

6 Call Count Variables

6.1 Call Counters

The Ingate firmware has 1 ***variable*** for use in GHM or cURL expressions which provide currently active call counts to the currently processing destination. It must be enclosed in `$(…)`.

Variable [URI.portion]	Explanation
`count.calls_to_this_user`[1]	An integer value of how many sessions active with this username as destination, identical to the username found in the To: header of the current request
[1] *(Ingate >= 6.2.2)*	

It is not flexible; you cannot do `$(count.john)`, but only `$(count.calls_to_this_user)`. To be clear: If the currently processing request (e.g. the incoming INVITE) is processed in the dialplan, and the call is to `john`, and there are already 2 other calls which go to `john@...`, then `$(count.calls_to_this_user)` will return 2. So this can be used to get (and, to send via curl) the number of calls to a *username*.

Example curl expression:

```
sip:$curl1(counter.php?did=$(to.user)&cc=$(count.calls_to_this_user))
```

The REST API server responds either with the call destination, or a cause code to terminate the call.

The back-end web-service looks in a table for the given DID and checks whether there are still available channels to allow the call, picks the destination and responds with a plain text sip destination.

6.2 Call Count Logic

The Ingate firmware has logical comparison ***variables*** for use in GHM, and more specifically CRE, which return TRUE or FALSE, based on an integer comparison with a call count. With this in mind, they must be used within a CT, e.g. `CONDIF`. They must be enclosed in `$(…)`.

Variable [URI.portion]		Explanation
`count.calls_to_this_user`[1]	`.lt.`*x*	Less than *x*
	`.le.`*x*	Less than or equal to *x*
	`.eq.`*x*	Equal to *x*
	`.ge.`*x*	Greater than or equal to *x*
	`.gt.`*x*	Greater than *x*
[1] *(Ingate >= 6.3.2)*		

These will also work with future variables which start with `count.` in case such are implemented.

Examples in context:

```
$(CONDIF.count.calls_to_this_user.lt.99)$(CONDYES…)$(CONDNO…)
$(CONDIF.count.calls_to_this_user.le.17)$(CONDYES…)$(CONDNO…)
$(CONDIF.count.calls_to_this_user.eq.0)$(CONDYES…)$(CONDNO…)
$(CONDIF.count.calls_to_this_user.ge.21)$(CONDYES…)$(CONDNO…)
$(CONDIF.count.calls_to_this_user.gt.98)$(CONDYES…)$(CONDNO…)
```

Full example:

```
sip:user@mycorp.com?From=%3csip%3a$(CONDIF.count.calls_to_this_user
.eq.0)$(CONDYES.PLAIN.33)$(CONDNO.PLAIN.44)%40nowhere.com%3e
```

This outputs `From: <sip:33@nowhere.com>` if there are 0 calls to the current callee, otherwise it outputs `From: <sip:44@nowhere.com>`

Full example:

```
sip:conference@$(CONDIF.count.calls_to_this_user.gt.20)$(CONDYES.PL
AIN.192.168.1.11)$(CONDNO.PLAIN.192.168.1.10)
```

This outputs `sip:conference@192.168.1.11` if there are more than 20 calls to the current callee, otherwise it outputs `sip:conference@192.168.1.10`

Full example:

```
$(CONDIF.count.calls_to_this_user.ge.1)$(CONDYES.PLAIN.sip:voicemai
l@pbx1.lan)$(CONDNO.PLAIN.sip:joe@pbx1.lan)
```

This outputs `voicemail@pbx1.lan` if there is 1 or more calls to the current callee, otherwise it outputs `joe@pbx1.lan`

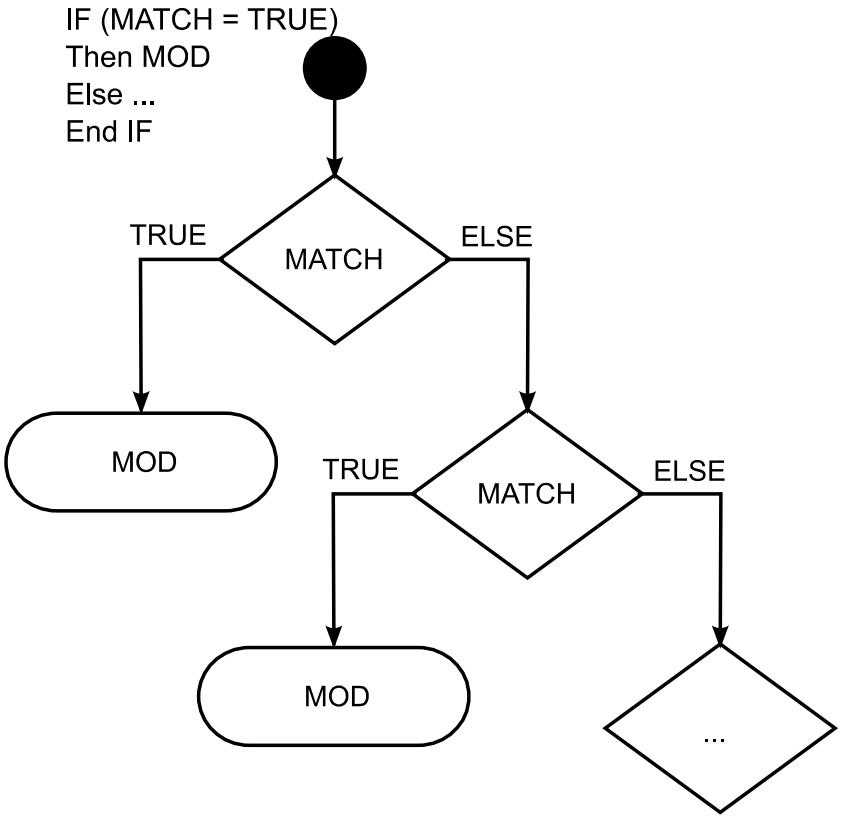
7 Conditionals

7.1 Conditional Regular Expressions (CRE)

CREs are a form of logic flow, where you can do for example:
If…*x* …is true, modify *x* to… *y*
Else if…*q*…is true, modify *q* to… *r*
On the value taken from *header*(*.portion*) at ingress.

Roughly: IF MATCH, then MODIFY, ELSE if match, then MODIFY, ELSE etc… …END.

This is the simplified general logic structure of a CRE:



The expressions are enclosed in `$()` – they behave as HAV, i.e. provide output for use in a GHM e.g. `$0?From=$(REGMATCH_123_...` which must be URI encoded and contain `%40` (i.e. an `@` symbol). See later examples

The following CRE:

```
...$(REGMATCH_123_REGMOD_456_REGELSE_(.*)_REGMOD_789_REGEND.from.user)...
```

first reads the `user` portion of the `From` header and then performs the regular match/replace which must be enclosed by the tags `REGMATCH_` and `_REGEND` as above. Explanation:

- Looks for 123 in the `user` portion of the `From` header. If the user portion contains 123, then the result is modified to 456.
- else, tries to greedily match anything (.*) against the `user` portion of the `From` header, so the result of the above expression would be 789.

This was a simplified example. In practice CRE can be much more complex.

7.1.1 Available statement keywords

CRE `$(action…)` statement keywords available for logic flow include:

```
$(REGMATCH_…
REGMOD_…
REGELSE_…
REGEND.…)
```

Note: `REGMOD`, `REGELSE` and `REGEND` statements cannot be orphaned, they must all be subsequent to a `REGMATCH` type statement.

7.1.1.1 The `REGMATCH` action statement

This is the start of your logic examination. This keyword tries to perform a match using a string or regular expression. E.g. to match the number string `123` or `456` you could use the statement:

```
$(REGMATCH_[0-9]{3}_…)
```

7.1.1.2 The `REGMOD` action statement

This statement outputs text based on a preceding `REGMATCH` statement that matched something from the input. It actually only outputs text. Its output values are used if the preceding `REGMATCH` matched. It can be augmented with e.g. `$1` if the `REGMATCH` captured something. To modify `jose` to `noway` you could use the expression:

```
$(REGMATCH_jose_REGMOD_noway_…)
```

7.1.1.3 The `REGELSE` action statement

This statement behaves identically to a `REGMATCH` statement, is subsequent to both a `REGMATCH` statement and a `REGMOD` statement e.g.

```
$(REGMATCH_[0-9]{3}_REGMOD_000_REGELSE_jose_REGMOD_noway_…)
```

Because logic is evaluated serially from left to right, you may want to ensure your logic flow starts more specifically/uniquely and then becomes more general e.g.:

```
$(REGMATCH_123_REGMOD_bingo_REGELSE_[0-9]{3}_REGMOD_dingo_…)
```

7.1.1.4 The `REGEND` action statement

This statement terminates a CRE analysis and is sibling to a header or its portion from ingress. E.g.

```
$(REGMATCH_[0-9]{3}_REGMOD_000_REGEND.from.user)
```

The above statement looks for a three-digit string in the user portion of the from header. Once your statement is closed, i.e. `REGEND`, if there are matches from ingress input, the `$(statement…)` provides output.

7.1.2 Worked Examples

7.1.2.1 Sweden E164 Number Normalization

The following expression will perform E164 normalization based on the `From` header of a telephone number (i.e. user portion) for Sweden:

- If the number starts with (^) a + followed by only digits until its end ($), leave the + and digits as is
- If the number starts with (^) 00 followed by only digits until its end ($), replace 00 with + and leave the digits as they are
- if the number starts with (^) 0 followed by only digits until its end ($), add +46 followed by the digits after the initial 0
- else, supply the number unmodified

The following is one continuous string:

```
...$(REGMATCH_^\+([0-9]+)$_REGMOD_+$1_REGELSE_^00([0-9]+)$_REGMOD_
+$1_REGELSE_^0([0-9]+)$_REGMOD_+46$1_REGELSE_(.*)_REGMOD_$1_REGEND
.from.user)...
```

Note: The result from the match from each `REGMATCH_` tag respective `REGELSE_` tag is valid only for a replacement in the respectively immediately following `REGMOD_` tag.

7.1.2.2 USA E164 Number Normalization

The following expression will perform E164 normalization based the `From` header of a telephone number (i.e. user portion) for USA:

- If the number already starts with a + followed by only digits, leave the + and digits as they are
- If the number starts with 001 followed by only digits, replace the 001 with +1 and leave the digits as they are
- if the number starts with 1 followed by only digits, add +1 followed by the digits after the initial 1

```
...$(REGMATCH_^001([0-9]{10})$_REGMOD_+1$1_REGELSE_^1([0-9]{10})$_REGMOD_+1$1_R
EGELSE_([0-9]{10})$_REGMOD_+1$1_REGEND.from.user)...
```

7.1.2.3 Generic Normalization into a + prefixed 8-12 digit phone number string

Rule

Order	*Look for the Prefix…*	*…and change it (add prefix) to*
1st	00	+
2nd	0	+ (national) e.g. +31
3rd	+	+ (no change)

4th	*(anything)*	+

As a Regexp

Order	*Match (and capture)*	*Modify*
1st	^00([0-9]{8,12})$	+$1
2nd	^0([0-9]{8,12})$	+31$1
3rd	^+([0-9]{8,12})$	+$1
4th	^(.*)$	+$1

Expression

	Match		*Modify*	
$(REGMATCH_	^00([0-9]{8,12})$	_REGMOD_	+$1	_REGELSE_
	^0([0-9]{8,12})$	_REGMOD_	+31$1	_REGELSE_
	^+([0-9]{8,12})$	_REGMOD_	+$1	_REGELSE_
	^(.*)$	_REGMOD_	+$1	_REGEND)

Resulting expression which operates on `user` portion of the `From` header

```
$(REGMATCH_^00([0-9]{8,12})$_REGMOD_+$1_REGELSE_^0([0-9]{8,12})$_RE
GMOD_+31$1_REGELSE_^+(.*)$_REGMOD_+$1_REGELSE_^(.*)$_REGMOD_+$1_REG
END.from.user)
```

The URI Encoded expression which produces a valid `From` header, taking from the `user` portion of `From` header at ingress:

```
?From=%3csip%3a$(REGMATCH_^00([0-9]{8,12})$_REGMOD_+$1_REGELSE_^0([
0-9]{8,12})$_REGMOD_+31$1_REGELSE_^+(.*)$_REGMOD_+$1_REGELSE_^(.*)$
_REGMOD_+$1_REGEND.from.user)%40$(from.host)$([from.uriparams])%3e$
(from.params)
```

Worked Example Results

Phone #	*…is modified to…*	*…due to*	*reason*
112	+112	^(.*)$ → +$1	3 digits, no 00, 0 or +
13009865555	+13009865555	^(.*)$ → +$1	11 digits, no 00, 0 or +
9865555	+9865555	^(.*)$ → +$1	7 digits, no 00, 0 or +
+9865555	++9865555	^(.*)$ → +$1	7 digits
03009865555	+313009865555	^0([0-9]{8,12})$ → +31$1	10 digits, with 0 prefix
008009865555	+318009865555	^00([0-9]{8,12})$→ +31$1	11 digits with 00 prefix

To avoid `112 → +112` you can modify the expression to:

```
$(REGMATCH_^00([0-9]{8,12})$_REGMOD_+$1_REGELSE_^0([0-9]{8,12})$_RE
GMOD_+31$1_REGELSE_^+(.*)$_REGMOD_+$1_REGEND.from.user)
```

i.e.

```
?From=%3csip%3a$(REGMATCH_^00([0-9]{8,12})$_REGMOD_+$1_REGELSE_^0([
0-9]{8,12})$_REGMOD_+31$1_REGELSE_^+(.*)$_REGMOD_+$1_REGEND.from.us
er)%40$(from.host)$([from.uriparams])%3e$(from.params)
```

Ingress number strings not 8-12 digits long will not be prefixed with a +, however.

Note: In firmware versions <= 5.0.11 you cannot use `$1?From=$(...$1...)` – i.e. capture groups on the trunk page (i.e. where `$1` is before *and* after the "`?`"), *and* `$REGMATCH` expressions which also contain `$1`. The workaround is to use `$0?From=$(....$1…)`. See Errata.

7.1.2.4 Forward To based on body content

The following expression in the Forward-to field of the dial-plan:

```
sip:$(REGMATCH_m=video_REGMOD_videouser@mycorp.com_REGELSE_.*_REGMO
D_audiouser@mycorp.com_REGEND.body.plain)
```

will forward to `videouser@mycorp.com` if the message body contains '`m=video`', otherwise it will forward to `audiouser@mycorp.com`

Note: requires firmware >= 6.2.0

7.2 Conditional Header Output (CHO) and Conditional Body Output (CBO)

Available: Ingate >= 5.0.4

A CHO Outputs Headers, or portions thereof, based on the evaluation, or validation, of Conditions. Similarly, a CBO Outputs the Body, or portions thereof.

CHO and CBO expressions are composed of one or more Condition Tests (CT) and one or more Conditional Results (CR) which perform Conditional Actions (CA) or provide one or more Conditional Output (CO).

CT – test for condition at ingress
CR – then enter a conditional branch result which
CA – perform conditional actions or
CO – provide conditional output

In its simplest form, a CHO is: CT → CR → CR:

`$(test.…)$(resultX.…)$(resultY.…)`

Their syntax is generally

`$(test.header.part)$(resultX.output.header.part)$(resultY.action)`
where `header` and `part` indicate the presence of URI or parts thereof at ingress.

`$(test.body.plain)$(resultX.action)$(resultY.action)`
where `body.plain` indicates body presence at ingress.

A simple CHO looks like this:

Conditional Header Output
`$(CONDIF.diversion.user)$(CONDYES.diversion.user)$(CONDNO.from.user)`

`$(CONDIF…)` test performs a Boolean evaluation of a condition. The result statements `$(CONDYES…)` and `$(CONDNO…)` are expressions which are subsequently evaluated and supply output, or perform an action depending on which condition holds true.

In the above CHO, when a `user` portion of a `Diversion:` header is present in an ingress SIP message (e.g. from the PBX), the `$(CONDIF…)` CHT test expression evaluates to **true**, i.e. the action **yes**, then the `$(CONDYES…)` CR is reached, the CHA action expression output supplies the `$(diversion.user)` header portion found at ingress, otherwise i.e. in the absence of a `user` portion of a `Diversion` header, the `$(CONDNO…)` CHA action expression supplies the `$(from.user)` portion of the ingress `from` header.

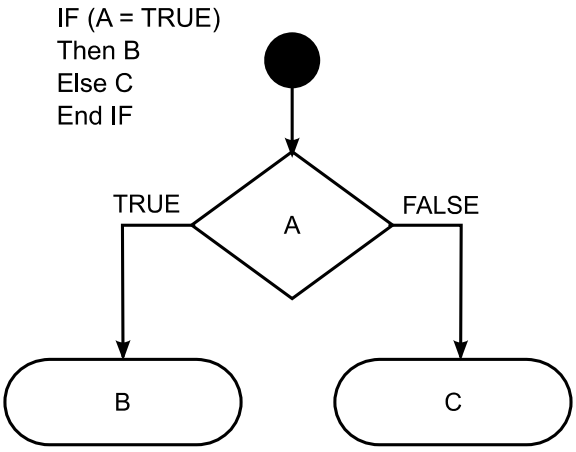
Note: This CHO expression evaluates to provide output, which is used to build a new header via a GHM. See later examples.

In older versions of this document – CHO were historically referred to as CHM (Conditional Header Manipulation).

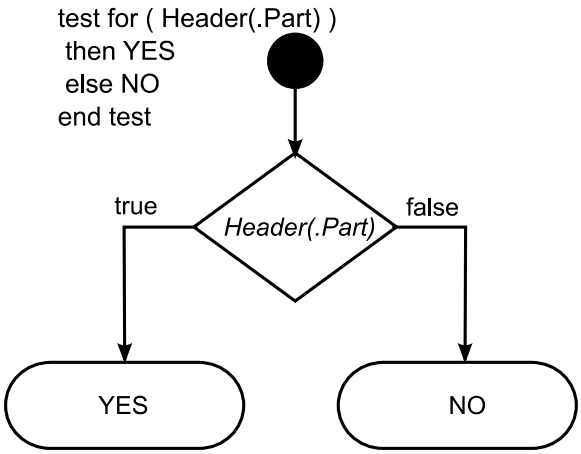
7.3 Conditional Test (CT)

CT statements Conditionally Test for headers or their parts. CT statement keywords are all Boolean: they test a condition and evaluate either to **true** or to **false**. CT can also contain a CRE (i.e. ….REGMATCH_ …), since a CRE can also either pass or fail i.e. evaluate to **true** or to **false**. See the section Conditional Regular Expressions (CRE).

This is a Boolean condition test:



This is how CT operate:



7.3.1 Available CT statement keywords

All operate in a Boolean fashion: They evaluate either to **true** or to **false** based on input conditions. All are case-sensitive: `Condif` is not synonymous to `CONDIF`.

CHT `$(test…)` keywords available for logic flow include:

```
$(CONDIF…)
$(CONDIFNOT…)
$(CONDANDIF…)
$(CONDANDIFNOT…)
$(CONDORIF…)
$(CONDORIFNOT…)
```

They all use the same construction hierarchy, and can contain CREs:

e.g.

```
$(CONDIF.REGMATCH_…)
$(CONDIFNOT.from.user…)
$(CONDANDIF.REGMATCH_…)
$(CONDANDIFNOT.diversion.domain…)
$(CONDORIF…)
$(CONDORIFNOT.REGMATCH_…)
```

Example:

```
sip:joe@$(CONDIF.REGMATCH_test_REGEND.from.user)$(CONDYES.PLAIN.192
.168.1.11)$(CONDNO.PLAIN.192.168.1.10)
```

Note: ANDIF(NOT) and ORIF(NOT) type statements cannot be orphaned, they must be subsequent to an IF(NOT) type statement to function correctly.

7.3.1.1 The CONDIF test statement

This statement; opens a CHO.

E.g. to evaluate whether a `user` portion of a `diversion` header was found at ingress:

```
$(CONDIF.diversion.user)
```

7.3.1.2 The CONDIFNOT test statement

This statement; also opens a CHO.

E.g. to evaluate whether a `P-Access-Network-Info` header was **not** found at ingress:

```
$(CONDIFNOT.P-Access-Network-Info)
```

7.3.1.3 The CONDANDIF test statement

This statement; continues a CHO; is *sub*ordinate to either `CONDIF` or `CONDIFNOT` statement.

E.g. to evaluate whether a `P-Called-Party-ID` header was found at ingress (in addition to prior logic flow):

```
$(CONDIF…)$(CONDANDIF.P-Called-Party-ID)
```

7.3.1.4 The CONDANDIFNOT test statement

This statement; continues a CHO; is *sub*ordinate to either `CONDIF` or `CONDIFNOT` statement.

E.g. to evaluate whether a `P-Visited-Network-ID` header was **not** found at ingress (in addition to prior logic flow):

```
$(CONDIFNOT…)$(CONDANDIFNOT.P-Visited-Network-ID)
```

7.3.1.5 The CONDORIF test statement

This statement; continues a CHO; is *sub*ordinate to either `CONDIF` or `CONDIFNOT` statement.

E.g. to evaluate whether a `P-Charging-Function-Addresses` header was found at ingress (in addition to prior logic flow):

```
$(CONDIF…)$(CONDORIF.P-Charging-Function-Addresses)
```

7.3.1.6 The `CONDORIFNOT` test statement

This statement; continues a CHO; is *sub*ordinate to either `CONDIF` or `CONDIFNOT` statement.

E.g. to evaluate whether a `P-Charging-Vector` header was **not** found at ingress (in addition to prior logic flow):

```
$(CONDIFNOT…)$(CONDORIFNOT.P-Charging-Vector)
```

7.4 Conditional Results (CR)

CT – test for condition at ingress
CR – then enter a conditional branch result which
CA – perform conditional actions or
CO – provide conditional output

CR statements Conditionally provide Results based on headers or their parts: they provide entry into a branch result. CR statements contain conditional actions – CA or provide conditional output – CO. CR are case-sensitive: `Condyes` is not synonymous to `CONDYES`.

7.4.1 Available CR keywords

CR `$(result…)` keywords available for logic flow include:

`$(CONDYES…)`
`$(CONDNO…)`

The following is a valid chain which provides multiple results based on a single condition:
`$(CONDYES…)$(CONDYES…)$(CONDYES…)$(CONDNO…)`

7.4.1.1 The CONDYES result statement

This statement is synonymous with the **true** condition

E.g. for a **true** condition:

`$(CONDIF…)$(CONDYES.action.…)`
`$(CONDIF…)$(CONDYES.output.…)`

7.4.1.2 The CONDNO result statement

This statement is synonymous with the **false** condition

E.g. for a **false** condition:

`$(CONDIF…)$(CONDNO.action.…)`
`$(CONDIF…)$(CONDNO.output.…)`

7.5 Conditional Actions (CA)

CT – test for condition at ingress
CR – then enter a conditional branch result which
CA – perform conditional actions or
CO – provide conditional output

CA statements perform Actions. CA provide zero output. CA are case-sensitive: `Abort` is not synonymous to `ABORT`. CA statements terminate a CR branch. CA statements are subordinate.

7.5.1 Available CA keywords

CR `$(…action…)` keywords available for logic flow include:

`$(…ABORT…)`

7.5.1.1 The `ABORT` action statement

E.g. to abort execution of a logic branch if a `user` portion of a `referred-by` header did not exist at ingress, i.e. no header? Abort by providing zero output:

```
$(CONDIF.Referred-By.user)$(CONDNO.ABORT)
$(CONDIFNOT.Referred-By.user)$(CONDYES.ABORT)
```

7.6 Conditional Output (CO)

CT – test for condition at ingress
CR – then enter a conditional branch result which
CA – perform conditional actions or
CO – provide conditional output

CO statements provide Output (which can also be zero-length). CO keywords are case-sensitive except for *header(.part)*: `Plain` is not synonymous to `PLAIN`. CO statements terminate a CR branch. CO statements are subordinate i.e. must follow CR.

7.6.1 Available CO keywords

CO `$(output…)` keywords available for logic flow include:

```
$(….PLAIN.…)
$(….header)
$(….header.part)
$(….body.plain)
```

CRE may also be used in a CO position e.g.

```
$(CONDIF.from.user)$(CONDYES.REGMATCH_…)
```

7.6.1.1 The `PLAIN` output statement

This statement outputs URI encoded plain text strings.
E.g.

```
$(CONDIF.from.user)$(CONDYES.PLAIN.%3csip%3a1234%40company.com%3e)
$(CONDIF.from.user)$(CONDNO.PLAIN.%3csip%3aABCD%40company.com%3e)
```

7.6.1.1 The *`header`* output statement

This statement outputs the named URI or header from ingress.
E.g.

```
$(CONDIF.from.user)$(CONDYES.from)
```

7.6.1.1 The *`header.part`* output statement

This statement outputs the named URI or header part from ingress.
E.g.

```
$(CONDIF.from.user)$(CONDYES.from.user)
```

7.7 Conditional Header Output (CHO) examples

7.7.1 Simple

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a1234%40company.com%3e)$(CONDNO.PLAIN.%3csip%3aABCD%40company.com%3e)
```

Explanation: If there is a `user` portion of a `Diversion:` header, then the `From:` header produced will be `<sip:1234@company.com>`, otherwise it will be `<sip:ABCD@company.com>`
Resulting in either:

(CONDYES) `...`

```
Diversion: joe@domain
From: <sip:1234@company.com>
```

or

(CONDNO) `...`

```
From: <sip:ABCD@company.com>
```

Note: headers of the format `<tel:…>` are accessed not with `header.user` but with `header.telnum`

7.7.2 More Complex Example

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a)$(CONDYES.REGMATCH_^\+([0-9]+)$_REGMOD_+$1_REGELSE_^00([0-9]+)$_REGMOD_+$1_REGELSE_^0([0-9]+)$_REGMOD_+46$1_REGELSE_(.*)_REGMOD_$1_REGEND.from.user)$(CONDYES.PLAIN.%40company.com%3e)$(CONDNO.PLAIN.%3csip%3aABCDEFGH%40company.com%3e)
```

This CHO does the following:

1. Where the ingress message contains a Diversion: header, then the username portion in the From: header (appended with “company.com”) replaces the From: header of the egress message, after applying Sweden E164 Number Normalization example.
2. Where the ingress message from the PBX does not contain a Diversion: header, no header manipulation is performed.

Resulting in either:

(CONDYES) `...`

```
Diversion: ext67@company.com
From: <sip:+46812345678@company.com>
```

or an unmodified from header:

(CONDNO) `...`

```
From: <sip:0812345678@company.com>
```

This Request at ingress…	…With this GHM…	…Becomes this request at egress
INVITE sip:zyx@ingate.com SIP/2.0 Session-Expires: 14400 Via: SIP/2.0/UDP 192.0.2.2:5060;branch=z9hG4bK4cc To: <sip:zyx@ingate.com> From: <sip:alpha@ingate.com>;tag=99 Call-ID: 21@sipgt-2d CSeq: 3 INVITE Contact: <sip:E4F0pr@192.0.2.2> Supported: timer, replaces, path, histinfo Allow: ACK, CANCEL, INVITE, BYE Max-Forwards: 15 Content-Type: application/sdp Content-Length: ...	?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a1234%40company.com%3e)$(CONDNO.PLAIN.%3csip%3aABCD%40company.com%3e)	INVITE sip:zyx@ingate.com SIP/2.0 Session-Expires: 14400 Via: SIP/2.0/UDP 192.0.2.2:5060;branch=z9hG4bK4cc72853 To: <sip:zyx@ingate.com> **From: <sip:ABCD@company.com>;tag=123** Call-ID: 21@sipgt-2d CSeq: 3 INVITE Contact: <sip:E4F0pr@192.0.2.2> Supported: timer, replaces, path, histinfo Allow: ACK, CANCEL, INVITE, BYE Max-Forwards: 15 Content-Type: application/sdp Content-Length: ...
INVITE sip:zyx@ingate.com SIP/2.0 Session-Expires: 14400 Via: SIP/2.0/UDP 192.0.2.2:5060;branch=z9hG4bK4cc Diversion: <sip:gohere@xcorp.xyz;reason=no-answer> To: <sip:zyx@ingate.com> From: <sip:alpha@ingate.com>;tag=55 Call-ID: 21@sipgt-2d CSeq: 3 INVITE Contact: <sip:E4F0pr@192.0.2.2> Supported: timer, replaces, path, histinfo Allow: ACK, CANCEL, INVITE, BYE Max-Forwards: 15 Content-Type: application/sdp Content-Length: ...	?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a1234%40company.com%3e)$(CONDNO.PLAIN.%3csip%3aABCD%40company.com%3e)	INVITE sip:zyx@ingate.com SIP/2.0 Session-Expires: 14400 Via: SIP/2.0/UDP 192.0.2.2:5060;branch=z9hG4bK4cc Diversion: <sip:gohere@xcorp.xyz;reason=no-answer> To: <sip:zyx@ingate.com> **From:<sip:1234@company.com>;tag=555665** Call-ID: 21@sipgt-2d CSeq: 3 INVITE Contact: <sip:E4F0pr@192.0.2.2> Supported: timer, replaces, path, histinfo Allow: ACK, CANCEL, INVITE, BYE Max-Forwards: 15 Content-Type: application/sdp Content-Length: ...

A note about parameters: we have not included any parameters in our expression. The b2bua will add its own special `;tag` parameter separately to the new From header. Any `;tag` parameter you copy from the source through the use of a `$(from.params)` expression will be updated separately by the B2BUA, but other parameters will remain unchanged. If you want to include any parameters found after `<sip:…>` in your result, add `$(from.params)` to your expression:

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a12345678%40company.com%3e)$(CONDNO.PLAIN.
%3csip%3aABCDEFGH%40company.com%3e)$(from.params)
```

7.7.3 More Complex Example - History-Info

Where forwarding information is transported via a History-Info: header, then the following should be used:

```
?from=$(CONDIF.history-info[-1].user)$(CONDYES.PLAIN.%3csip%3a)
$(CONDYES.REGMATCH_^\+([0-9]+)$_REGMOD_+$1_REGELSE_^00([0-9]+)$_RE
GMOD_+$1_REGELSE_^0([0-9]+)$_REGMOD_+46$1_REGELSE_(.*)_REGMOD_$1_R
EGEND.from.user)$(CONDYES.PLAIN.%40company.com%3e)
```

Note: that history-info[-1] refers to the last history-info header present in the SIP message from the PBX.
Resulting in either:

(CONDYES) . . .

```
History-Info: qwerty
From: <sip:+46812345678@company.com>
```

or an unmodified from header:

(CONDNO) . . .

```
From: <sip:0812345678@company.com>
```

7.7.4 More Complex Example – Conditional From Header, based on To callee

A company has an old PRI. Outbound calls are now made over the new SIP trunk. On standard calls, they want to send the main number of the PRI out over the SIP trunks. They need to send the real caller ID of the SIP trunks on emergency 911 calls, however.

The old PRI number is 555-101-2001.
Their new SIP trunk number is 555-777-8888.

So, if the call is to 911 i.e.
the user portion of the `To` header is 911,
the resulting `From` number is 5557778888,
if the call is not to 911 (i.e. to anything other than 911),
the resulting `From` number is 5551012001.

A suitable expression would be:

```
?from=$(CONDIF.REGMATCH_^911$_REGEND.to.user)$(CONDYES.PLAIN.%3csi
p%3a5557778888%40192.0.2.2%3e)$(CONDNO.PLAIN.%3csip%3a5551012001%4
0192.0.2.2%3e)
```

Resulting in either (911 is called):

(CONDYES) . . .

```
From: <sip:5556668888@192.0.2.2>
```

Or (any other number is called):

(CONDNO) . . .

```
From: <sip:5551012001@192.0.2.2>
```

7.8 URI Parameter Chaining

You can also manipulate multiple headers this way, by chaining the header manipulations in the usual way with the `&` character. The example below is like the previous `History-Info` example above, plus it sets the field `P-Preferred-Identity`:

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a)$(CONDYES.
REGMATCH_^\+([0-9]+)$_REGMOD_+$1_REGELSE_^00([0-9]+)$_REGMOD_+$1_R
EGELSE_^0([0-9]+)$_REGMOD_+46$1_REGELSE_(.*)_REGMOD_$1_REGEND.from
.user)$(CONDYES.PLAIN.%40company.com%3e)&P-Preferred-Identity=%3cs
ip%3aanother_id%40shop.com%3e
```

Resulting in either:

(CONDYES) . . .

```
Diversion: mike@domain
From: <sip:+46812345678@company.com>
P-Preferred-Identity: <sip:another_id@shop.com>
```

or an unmodified from header:

(CONDNO) . . .

```
From: <sip:0812345678@company.com>
P-Preferred-Identity: <sip:another_id@shop.com>
```

Note: If you want to add a header (which did not already exist in the message) only under certain circumstances e.g.: Replace a possibly existing `Referred-By`: header with a `Diversion`: header

```
?Diversion=$(CONDIF.Referred-By.user)$(CONDNO.ABORT)$(CONDYES.PLAI
N.%3csip%3a)$(CONDYES.Referred-By.user)$(CONDYES.PLAIN.%40)$(CONDY
ES.Referred-By.host)$(CONDYES.PLAIN.%3e)&Referred-By=__remove
```

If no Referred-By header is present, then without the `$(CONDNO.ABORT)` the following evaluation results:

```
?Diversion=
&Referred-By=__remove
```

which results in an empty diversion header in the resulting message. But with the `$(CONDNO.ABORT)`, the resulting header manipulation string becomes `?Diversion=$(ABORT)&Referred-By=__remove` and since header manipulations with unresolved variables will be skipped, no empty diversion header will be added. In effect becoming just:

```
?Referred-By=__remove
```

7.8.1 Illegal Chaining

The following example of chaining is illegal, and will not work:

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a&P-Preferre
d-Identity=%3csip%3aanother_id%40shop.com%3e)
```

For the above example to compile, it must be corrected – note the closing parenthesis:

```
?from=$(CONDIF.diversion.user)$(CONDYES.PLAIN.%3csip%3a)&P-Preferr
ed-Identity=%3csip%3aanother_id%40shop.com%3e
```

Summary: Parameter chaining only works outside of the regular expression `$()` context, and cannot be chained conditionally.

7.9 Keyword / Grammar and Syntax Summary for CHO and CRE

7.9.1 Tests

$(
[CONDIF(.REGMATCH…) |
CONDIFNOT |
CONDANDIF |
CONDANDIFNOT |
CONDORIF |
CONDORIFNOT].*Header*.*Portion*
)

7.9.2 Results

$(
CONDYES
.PLAIN.*Text* |
.*Header*.*Portion* |
.ABORT
| CONDNO
.PLAIN.*Text* |
.*Header*.*Portion* |
.ABORT
)

7.9.3 RegExp

$(
REGMATCH_
REGMOD
REGELSE
REGMOD
…
_REGEND.*Header*.*Portion*
)

Note: Statement keywords and expressions are evaluated from left to right, i.e. forwards in their construction.

8 Generic Header Manipulation (GHM)

The SIP header:

```
Diversion: <sip:7202839130@192.168.1.1>
```

Can be produced by the GHM:

```
?Diversion=%3csip%3a7202839130%40192.168.1.1%3e
```

GHM for egress Requests (`INVITE`, `REGISTER`, …) are invoked by `?`
GHM for egress Responses (`200 OK`, `180 Ringing`, …) are invoked by `?!`

Remove headers at egress by assigning the reserved value `__remove` (two underscores).

GHM are generally:

```
sip:URI?header=value
```

GHM to do multiple headers at once:

```
sip:URI?header=valueX&header2=valueY
```

And *`value`* can be composed of CHO, CRE, HAV, etc i.e.:

```
sip:user@host?header=CHO&header2=CRE
```

Characters disallowed or reserved in RFC 3261 must be escaped with a `%HEX` notation where `HEX` is a 2-digit hexadecimal number representing the escaped character (i.e. URI-Encoded String).

The table below lists often used characters, and their corresponding URI encoded HEX value:

Character	HEX Value
@	%40
:	%3a
;	%3b
,	%2c
=	%3d
<	%3c
>	%3e
Spaces	+ or %20

Note: < and > are required in the Header Value under some conditions. Consider it best practice to always use them. Refer to RFC 3261 section 20.10.

A fully constructed expression used in a *Forward To* might be:

```
sip:request-URI;tag1;tag2?header=value&header2=value2!header=value
```

Request-URI	GHM separator & header name	%HEX URI-Encoded header value
sip:$1@192.168.1.1	?Diversion=	%3csip%3a72839130%40192.168.1.1%3e

8.1 GHM for Requests(?...&...)

8.1.1 Adding or Replacing Headers

GHMs for Request methods (INVITE, REGISTER, OPTIONS, …) are denoted by the character ?

For GHM:

-If a specified URI exists at ingress, the header at egress is replaced.
-If a specified URI is absent at ingress, the header at egress is added.

```
?header=%3cUri-encoded_string%3e&header2=%3cUri-encoded_string%3e
```

8.1.2 Examples: GHM for Requests(?): Adding or Replacing Headers

Header	GHM strings which produce it
P-Asserted Identity	sip:$1@192.168.1.1?P-Asserted-Identity=%3csip%3a7202839130%40192.168.1.1%3e
	sip:$1@example.com?P-Asserted_Identity=%3csip%3a7202839130%40192.168.1.1%3e
Diversion	sip:$1@192.168.1.1?Diversion=%3csip%3a7202839130%40192.168.1.1%3e
Privacy	sip:$1@example.com?Privacy=<url-encoded_string>
Multiple headers	sip:$1@192.168.1.1?P-Asserted-Identity=<url-encoded>&Diversion=<url-encoded_string>
with tags	sip:$1@192.168.1.1;b2bua;from="Anonymous@10.182.0.178"?P-Asserted-Identity=<url-encoded_string>&Privacy=id
Replace the Allow header (Ingate <=5.0.6)	?Allow=ACK%2cINVITE%2cBYE%2cCANCEL%2cOPTIONS

Note: Replace %3cUri-encoded_string%3e with a valid URI-encoded header value.

8.1.3 Removing headers

```
?header=__remove&otherheader=__remove
```

Headers specified for removal which exist at ingress, are removed at egress.

8.1.4 Examples: Removing a header

Removing Privacy Header
`sip:$1@192.168.1.1?Privacy=__remove`
`sip:$1@example.com?P-Asserted-Identity=__remove`
`sip:$(ruri.user)@$(ruri.host)?Diversion=__remove`

8.1.5 Indices, Indexes, [?]; limiting the scope of operation

`?History-Info=__remove`	removes all `History-Info` headers
`?History-Info[2]=__remove`	removes only the second `History-Info` header
`?History_Info=something`	sets all `History-Info` headers to the same value `something`
`?History_Info[2]=something`	sets only the 2nd one to `something`, the 1st and 3rd remain unchanged
`?History_Info=__remove&History_info=something`	removes all `History-Info` headers and only one new `History-Info` header with value `something` is created.

See also Indices, indexes, [?]

8.2 GHM for Responses(?!...&!...)

8.2.1 Adding or Replacing Headers

GHM for egress Responses (`200 OK`, `180 Ringing`, …) are invoked by `?!`

```
?!header=%3cUri-encoded_string%3e&!header2=%3cUri-encoded_string%3e
```

Example solution to Avaya display problem using `P-Asserted-Identity`:

`sip:$1@192.168.1.1?!P-Asserted-Identity=$(to.dnameuri)`	Add P-Asserted Identity Header to all egress responses

This adds `P-Asserted-Identity` to the response, with the display name and URI taken from the `To` header at ingress.

8.2.2 Removing headers

`?!`*`header`*`=__remove&!`*`otherheader`*`=__remove`

8.3 GHM for Requests (?...&…) and Responses(?!...&!) combined in one expression

8.3.1 Adding or Replacing Headers

`?!`*`response-hdr01`*`=%3c…%3e&!`*`response-hdr02`*`=%3c…%3e&`*`req-hdr01`*
`=%3c…%3e&`*`req-hdr02`*`=%3c…%3e`

8.3.1 Removing headers

`?`*`req-header`*`=__remove&!`*`response-header`*`=__remove`

8.4 Multiple Occurrences of the same Header

Header fields are indexed using angle brackets [x] so that one can refer to the *n*:th occurrence of any header.

8.4.1 Breakout Example

The following is just one long line:

```
sip:$0@example.com?User-Agent=$(hdr.user-agent)
&Contact=sip%3afoo%40$(ip.eth4)%3buser%3dphone
&Organization=mycompany&Privacy=__remove
&Diversion[1]=sip%3a$(diversion[1].user)%401.1.1.1
```

Explanation of each Reg Expr component:

Component	Explanation
`sip:$0@example.com`	This is the Request URI, `$1` expresses the first variable captured in, e.g. from *Matching Request URI* field in the SIP Traffic – Dial Plan page.

?User-Agent=$(hdr.user-agent)	User-Agent header is supplied with the value taken from the ingress User-Agent header, where it exists. In B2BUA mode, the Ingate replaces the User-Agent header with its own string.
&Contact=sip%3afoo%40$(ip.eth4)%3buser%3dphone	The Contact Header is replaced with the IP address of eth4
&Organization=mycompany	The header Organization is added
&Privacy=__remove	The Privacy header is removed.
&Diversion[1]=sip%3a$(diversion[1].user)%401.1.1.1	The first Diversion header found, a new host portion is entered as 1.1.1.1.

8.5 Header Access Variables

Header Access Variables can be used in GHM as for Regular Expressions. The pre-defined variables can be used in GHM expressions. HAV provide read-only values of headers, or portions thereof found at ingress, in the result. See the chapter Header Access Variables for a list of available variables.

8.5.1 Header Access Variable Examples

Expression	Explanation
sip:$0@example.com?Contact=sip%3afoo%40$(ip.eth3)%3buser%3dphone	Modify Contact header with eth3 IP address
sip:$0@192.168.1.2?from=%3Csip%3a%2B$(from.user)%40172.16.0.1%3e	Add special character + to the from header
sip:$0?From=%3Csip%3a$(REGMATCH_^001([0-9]{10})$_REGMOD_+1$1_REGELSE_^1([0-9]{10})$_REGMOD_+1$1_REGELSE_([0-9]{10})$_REGMOD_+1$1_REGEND.from.user)%40$(from.host)$(CONDIF.from.uriparams)$(CONDYES.from.uriparams)%3e$(from.params)	Reduce all variants of the from header prefix 01, 1, or +1 to +1, pass any other URI parameters
sip:$0?From=%3Csip%3a$(REGMATCH_^001([0-9]{10})$_REGMOD_+1$1_REGELSE_^1([0-9]{10})$_REGMOD_+1$1_REGELSE_([0-9]{10})$_REGMOD_+1$1_REGEND.from.user)%40$(from.host)$([from.uriparams])%3e$(from.params)	Reduce all variants of the from header prefix 01, 1, or +1 to +1, pass any other URI parameters (alternative)
sip:$1@192.168.1.1?From=$(from.uri)	Strip the Display name portion out of the from header
+32$1?Diversion=+$(Diversion.telnum)	prefix + and re-write an ingress tel: format Diversion header to simply +...

9 Supplementary examples from real-world support cases

-Document ends-