ACOS 6.0.7
aFleX Scripting Language Reference

May, 2025

Information in this document is subject to change without notice.

PATENT PROTECTION

A10 Networks, Inc. products are protected by patents in the U.S. and elsewhere. The following website is provided to satisfy the virtual patent marking provisions of various jurisdictions including the virtual patent marking provisions of the America Invents Act. A10 Networks, Inc. products, including all Thunder Series products, are protected by one or more of U.S. patents and patents pending listed at: a10-virtual-patent-marking.

TRADEMARKS

A10 Networks, Inc. trademarks are listed at: a10-trademarks

CONFIDENTIALITY

This document contains confidential materials proprietary to A10 Networks, Inc. This document and information and ideas herein may not be disclosed, copied, reproduced or distributed to anyone outside A10 Networks, Inc. without prior written consent of A10 Networks, Inc.

DISCLAIMER

This document does not create any express or implied warranty about A10 Networks, Inc. or about its products or services, including but not limited to fitness for a particular use and non-infringement. A10 Networks, Inc. has made reasonable efforts to verify that the information contained herein is accurate, but A10 Networks, Inc. assumes no responsibility for its use. All information is provided "as-is." The product specifications and features described in this publication are based on the latest information available; however, specifications are subject to change without notice, and certain features may not be available upon initial product release. Contact A10 Networks, Inc. for current information regarding its products or services. A10 Networks, Inc. products and services are subject to A10 Networks, Inc. standard terms and conditions.

ENVIRONMENTAL CONSIDERATIONS

Some electronic components may possibly contain dangerous substances. For information on specific component types, please contact the manufacturer of that component. Always consult local authorities for regulations regarding proper disposal of electronic components in your area.

FURTHER INFORMATION

For additional information about A10 products, terms and conditions of delivery, and pricing, contact your nearest A10 Networks, Inc. location, which can be found by visiting www.a10networks.com.

Table of Contents

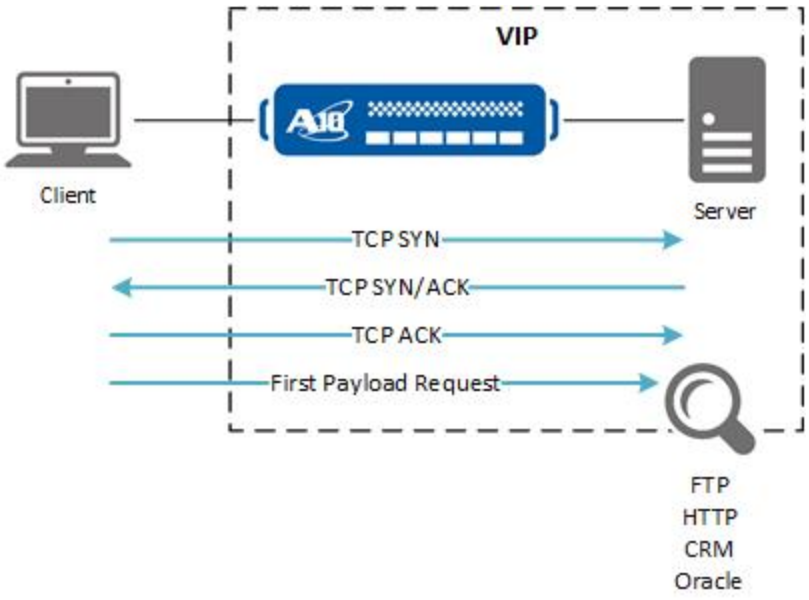
Getting Started

The aFleX scripting language is a powerful inline custom scripting engine that provides in-depth, granular control of inspection and redirection policies (filter, drop, redirect). The aFleX scripting language is based on the Tool Command Language (Tcl) programming standard for simplicity and familiarity. For an aFleX policy to work, it must be bound to a virtual port on the ACOS device. Then the aFleX policy can make policy decisions by inspecting the payload packets from all the traffic going through the virtual port.

Below is an example of a simple aFlex script:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.1] } {
        pool www_service_group
    }
}
```

Figure 1 : aFleX overview



The chapters provide detailed information about working with aFleX policies.

The following topics are covered:

Advantages of Using aFleX Policies

- aFleX policies allow you to exercise more granular control of packet inspection and traffic load balancing.
- aFleX policies can redirect traffic to a group of servers that are bound to a virtual port, to one specific server in a pool (service group), or to individual ports and URIs on a specific pool member (server).
- aFleX policies provide complete flexibility supporting both simple and sophisticated content-switching needs.
- aFleX policies can search packet headers or even the actual packet content, and direct packets based on the search results.
- aFleX policies can maintain persistence.
- Tcl scripts created using competitors' scripting engines often can be easily converted into aFleX scripts, providing backwards compatibility for customized solutions.
- aFleX policies are powerful with advanced functions. aFleX policies can be used as a security tool and even provide a quick solution for zero-day vulnerability mitigation.

aFleX Processing Order

aFleX policies have higher priority than most templates, except cookie persistence templates. The complete SLB processing order for virtual port traffic is mentioned in the following topics.

The following topics are covered:

Packet Processing Order for Layer 4 Virtual Ports

For Layer 4 virtual ports (TCP, UDP, or “Others”), template parameters are processed in the following order:

1. aFleX policy (script)
2. DNS template
3. Policy template
4. All other types of templates

Packet Processing Order for Layer 7 Virtual Ports

1. For Layer 7 virtual ports (for example: HTTP), template parameters are processed in the following order:
2. Layer 4 packet processing (described above in Packet Processing Order for Layer 4 Virtual Ports)
3. Layer 7 server selection:
    a. Cookie persistence template
    b. aFleX policy (script)
    c. All other types of templates

Packet Processing Example

A virtual port is bound to an aFleX policy and two application templates, a URL switching template and a cookie persistence template.

Both the URL switching template and the aFleX policy are applicable to a client’s traffic. The URL switching template chooses server server10, but the aFleX policy chooses another server, server20. Since the aFleX policy has higher priority, the traffic is directed to server20. However, if the cookie persistence template selects server30, the traffic ultimately will be directed to server30.

NOTE: Server template limits are applied for both service-group and server selection. Commands that call for server selection (i.e., “node”, “pool”, “persist”, etc.) will enforce server template limits on the selected server. As a result, new connections that match a persist uie entry may be unable to use the rport and a default server selection will occur instead. To prevent default server selection, use the **`def-selection-if-pref-failed-disable`** command for the vport.

When aFleX Policy Changes Take Effect

aFleX policy changes do not affect traffic that is already active on a virtual port. For example, if you bind an aFleX policy to a virtual port on which some traffic sessions are already active, the aFleX policy does not affect those sessions. The aFleX policy only affects sessions that begin after the aFleX policy is applied to the virtual port. Likewise, if you change an aFleX policy that is already bound to a virtual port, the changes do not apply to the sessions that are active when you change the policy. The active sessions are still processed using the aFleX policy as it was before the changes. The policy changes apply only to the sessions that begin after the policy changes are saved.

Support for Multiple aFleX Policies on a Single Virtual Port

You can bind up to 16 aFleX scripts to be bound to a single virtual port. When multiple aFleX scripts are bound to a virtual port, the scripts are processed from top down beginning with the first script bound to the virtual port and ending with the last script bound to the virtual port. The multiple scripts are processed exactly as if they were concatenated together into a single aFleX script. Multiple events of the same type are executed sequentially (top to bottom), as though they were all in the same script.

Configure aFleX for GTP Director

The aFleX infrastructure is used to configure the GTP Director and check the GTP packets. aFleX redirects the traffic to a specific home PGW based on the GTP

payload. The aFleX ruleset can be updated as per the requirement of the organization. The sections mentioned below include information about configuring aFleX for GTP Director.

The following topics are covered:

Ruleset for Defining Payload

The Ruleset defines the criteria to match the GTP payload for redirecting the traffic to specific Gateway or Server or Service-Group.

Example Configuring ruleset on ACOS device:

Rule1: If IMSI starts with '466924' and APN starts with 'internet' then direct to SG1.

Rule 2: If IMSI starts with '466777' and APN starts with 'internet' then direct to SG2.

Rule 3: If IMSI starts with '355000' and APN starts with 'pan' then direct to SG3.

Syntax to Define Ruleset

```
RuleNum#FirstOption-Matchtype-Value:SecondOption-Matchtype-
Value:ThirdOption-Matchtype-Value Service-Group
```

Table 1 : Match Type

c	Contains
s	Starts_with
e	Ends_with
q	Equals

For separating the conditions, use the following symbols:

- Conditions— ':'
- Parameter— '-'
- Service-group— ' ' (space)

The ruleset for match type is added in the following aFleX example:

```
set::RuleSets {
 1:imsi-e-3930:apn-s-int:pdna-q-0.0.0.0:sg2
 71:mei-e-4916:apn-s-int:pdn-q-3:sg2
 72:mei-e-4916:apn-s-int:ambruplink-q-150000:ambrdownlink-q-800000:sg2
 75:mei-e-4916:apn-s-int:pdn-q-3:sg2
 76:imsi-e-3930:apn-s-int:msisdn-e-066821:sg2
 79:mei-e-4916:apn-s-int:pdna-q-0.0.0.0:sg2
 82:imsi-e-3930:apn-s-int:rat-q-6:sg2
 83:imsi-e-3930:apn-s-int:mcc-e-440:mnc-e-10:sg2
 86:imsi-e-3930:apn-s-int:tac-e-85:tcellid-e-641:sg2
 87:imsi-e-3930:apn-s-int:fteidkey-e-4eeb:fteid-q-49.103.66.36:sg2
 95:imsi-e-3930:apn-s-int:cc-e-a00:sg2
 99:mei-e-4916:apn-s-int:pdn-q-3:sg2
 }
```

Configuring aFleX for GTP Director

To configure the aFleX GTP Director, perform the following steps:

1. On the ACOS device, increase the system resource-usage max-aflex-file-size to 128

   ```
   ACOS(config)#system resource-usage max-aflex-file-size 128
   ```

2. Import or create aFleX.
   a. Create aFleX using CLI.

      In the configuration mode, perform the following steps:

      i. Enter the command:

      ```
      aflex create aflex_gtp
      ```

      ii. Copy and paste the aFlex and insert '. ' (Dot) at the end

      Example:

```
ACOS(config)#aflex create test1
```

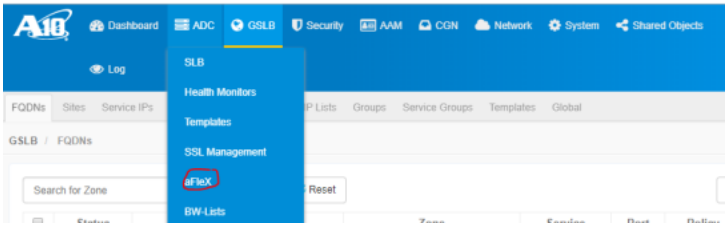
NOTE: Type in your aFleX script (type '.' on a line by itself when done)

```
when HTTP_REQUEST {
  HTTP::redirect https://[HTTP::host][HTTP::uri]
}
.
aFleX test1 created; syntax check passed
ACOS(config)#
```

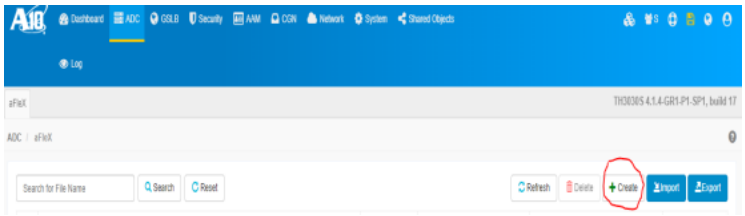
b. Create aFleX using GUI.

i. Login

ii. Click ADC >> aFlex

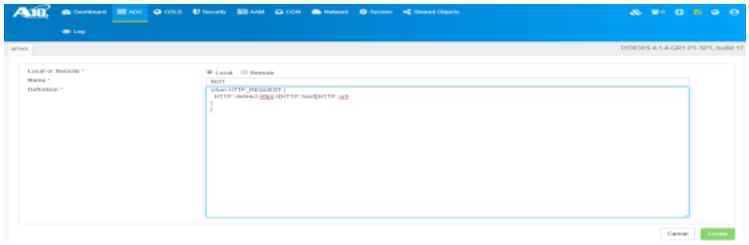


iii. Click Create

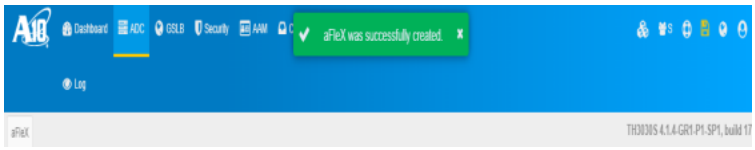


iv. In the Name box, enter a name

v. In the Definition box, add the aFleX script



vi. Click Create



The following fields are extracted from the GTP packet:

Table 2 : GTP Packet

SL #	Field	Match Keywords	Key	IE Type	Values	Example	As Shown in Wireshark
1	IMSI (International Mobile Subscriber Identity)	Match the IMSI number	imsi	1	decimal	4.66924E+14	IMSI: 466924000003930
2	APN (Access Point Name)	Match field APN.	apn	71	Alpha-numeric	Internet.mnc092.mcc466.gprs	APN (Access Point Name): internet.mnc092.gprs
3	MSISDN (Mobile Subscri	Match field	msisdn	76	decimal	8.86933E+11	E.164 number (MSISDN): 886933066821

Table 2 : GTP Packet

SL #	Field	Match Keywords	Key	IE Type	Values	Example	As Shown in Wireshark
	ber ISDN number)	MSISDN					
4	MEI (ME Identity)	Match field MEI	mei	75	decimal	3.53043E+15	MEI(Mobile Equipment Identity): 3530430928044916
5	ULI (User Location Info)	TAC – Tracking Area Code	tac	86	decimal	5985	TAC (Tracking Area Code): 0x1761 (5985)
		E-UTRAN Cell Identifier.	tcellid		decimal	50074641	ECI (E-UTRAN Cell Identifier): 50074641
6	MNC (Mobile Network Code)	MCC (Mobile Country Code)	mnc	83	decimal	10	Mobile Network Code (MNC): NTT DOCOMO, INC. (10)

Table 2 : GTP Packet

SL #	Field	Match Keywords	Key	IE Type	Values	Example	As Shown in Wireshark
7	MCC (Mobile Country Code)	MNC (Mobile Networks Code)	mcc	83	decimal	440	Mobile Country Code (MCC): Japan (440)
8	RAT (Radio Access Technology)	Match field RAT-Type	rat	82	decimal	6	RAT Type: EUTRAN (6)
9	CC (Charging Characteristics)	Match field CC, the hex number.	cc	95	Hex	0x0a00	0000 1010 0000 0000 = Charging Characteristic: 0x0a00
10	PDN Type	value	pdn	99	decimal	3	PDN Type: Ipv4/IPv6 (3)
11	PDN Address	Ipv4 and ipv6 address	pdna	79	IP format	0.0.0.0 - IPv4	PDN Address and Prefix(IPv6) 00000000000000000000000000000000

Table 2 : GTP Packet

SL #	Field	Match Keywords	Key	IE Type	Values	Example	As Shown in Wireshark
		match					PDN Address and Prefix(IPv6): 0.0.0.0
12	F-TEID (Fully Qualified Tunnel End-point ID)	TEID/GRE Key F-TEID IPv4	fteidkey fteid	87	Hex Ip Format	001a4eeb 49.103.66.36	TEID/GRE Key: 0x001a4eeb F-TEID IPv4: 49.103.66.36
13	APN-AMBR (Aggregate Maximum Bit Rate)	Match Uplink, downlink	ambruplink ambrdownlink	72	Decimal	150000 800000	AMBR Uplink (Aggregate Maximum Bit Rate for Uplink): 150000 AMBR Downlink (Aggregate Maximum Bit Rate for Downlink): 800000

Add aFleX to the virtual port of the virtual server

```
slb virtual-server vip1 200.200.200.200
                port 2123 udp
         service-group sg1
                gtp-session-lb
                aflex gtp_proxy << Bind the aFlex
!
```

NOTE:

- Only GTPv1-C and GTPv2-C is supported.
- The rule set defined is not in preferential order. However, the match is done in the order in which the ruleset is configured and the first match is chosen
- Up to 64 sets of match criteria are supported.
- Up to 16 service-groups are supported and if none of the configured set of Match criteria is matched, then the default service-group configured under the VIP is used to select one of the PGW's.

The following fields are extracted and can be used to match the GTPv1 requests.

Table 3 : GTP Requests Match

SL #	Field	Match Keyword		IE Type	Values	Example	As shown in Wireshark
1	International Mobile Subscriber Identity (IMSI)	Match the IMSI number	imsi	2	decimal	imsi-q-466924100000001	IMSI - 466924100000001
2	Routeing Area Identity (RAI)	Routeing Area Identity codes	mcc mnc lac rac	3	decimal	mcc-q-466 mnc-q-92 lac-q-65534 rac-q-255	Mobile Country Code (MCC): Unassigned (466) Mobile Network Code (MNC): Unknown (92) Local Area Code (LAC):

Table 3 : GTP Requests Match

SL #	Field	Match Keyword		IE Type	Values	Example	As shown in Wireshark
							65534 Routing Area Code (RAC): 255
3	Selection mode	Match selection mode type	selection_mode	15	decimal	selection_mode-q-0	MS or network provided APN subscribed verified 00 = MS or network provided APN subscribed verified (0)
4	Tunnel Endpoint Identifier Data I	TEID I value	teid_data1	16	decimal	teid_data1-q-216487	0x7c4978a0 (2085189792)
5	Tunnel Endpoint Identifier Control Plane	TEID_CP value	teid_cp	17	decimal	teid_cp-q-168041424	0x7c4978a0 (2085189792)
6	Access Point Name	Match APN field.	apn	131	Alphanumeric	apn-q-TEST7L2EPG1	TEST7L2EPG1 APN Length: 12 APN: TEST7L2EPG1

Table 3 : GTP Requests Match

SL #	Field	Match Keyword		IE Type	Values	Example	As shown in Wireshark
7	GSN Address	GSN value	gsn_address	133	IP	gsn_address-q-221.120.91.205	221.120.91.205 GSN address length: 4 GSN address IPv4: 221.120.91.205
8	MS International PSTN/ISDN Number (MSISDN)	PSTN/ISDN number	msisdn	134	decimal	msisdn-q-886905000201	Length: 7 1... ...= Extension: No Extension .001 ....= Nature of number: International number (0x1) ..... 0001= Number plan: ISDN/Telephony Numbering (Rec ITU-T E.164) (0x1) E.164 number (MSISDN): 886905000201

Table 3 : GTP Requests Match

SL #	Field	Match Keyword		IE Type	Values	Example	As shown in Wireshark
							Country Code: Taiwan, China (886)
9	RAT Type	UTRAN	rat_type	151	decimal	rat_type-q-1	UTRAN Length: 1 RAT Type: UTRAN (1)
10	User Location Information	TAC – Tracking Area Code	sai mcc mnc lac_uli sac	152	decimal	sai-q-1 mcc-q-466 mnc-q-92 lac_uli-q-10703 sac-q-21724	Length: 8 Geographic Location Type: Service Area Identity (SAI) (1) Mobile Country Code (MCC): Unassigned (466) Mobile Network Code (MNC): Unknown (92) Location Area Code (SAC): 21724
11	IMEI(SV)	Internati	imei	154	decimal	imei-q-35472005366	3547200536635901

Table 3 : GTP Requests Match

SL #	Field	Match Keyword		IE Type	Values	Example	As shown in Wireshark
		onal Mobile Equipment Identity Number				35901	Length: 8 IMEI (SV): 3547200536635901

Define the GTPv1 ruleset in the following part of the code:

```
set::RuleSetsV1 {
        2:imsi-q-466924100000001:sg2
        3:mcc-q-466:mnc-q-92:lac-q-65534:rac-q-255:sg2
        15:selection_mode-q-1:sg2
        16:teid_data1-q-216487:sg2
        17:teid_cp-q-168041424:sg2
        131:imei-s-3547:apn-e-TEST7L2EPG1:msisdn-e-0201:sg2
        133:gsn_address-q-221.120.91.205:sg2
        134:msisdn-q-886905000201:sg2
        151:imei-s-3547:rat_type-q-1:msisdn-e-0201:sg2
        152:sai-q-1:lac_uli-q-10703:sac-q-21724:sg2
        154:imei-s-3547:rat_type-q-1:msisdn-e-0201:sg2
        }
```

Example:

The packet can contain IMSI, MEI, MCC, MNC. Another packet might contain IMSI, MEI, RAT.

aFleX CLI Commands

For information about importing and binding aFleX scripts, see Applying aFleX Scripts To Virtual Ports.

This section includes information about working with aFleX scripts in the CLI.

The following topics are covered:

aFleX Online Help

You can access aFleX help information at the global configuration level of the CLI.

NOTE: aFleX help information is available through the CLI only and not accessible from the GUI.

Summary of the aFlex help commands are mentioned in Table 4:

Table 4 : aFlex Help Commands

Command	Description
aflex help events	View help for aFleX events.
aflex help global	View help for aFleX global commands.
aflex help operators	View help for aFleX operators.
aflex help *command*	View help for a specific aFleX command.

This example displays help information for aFleX `TIME` commands:

```
ACOS(config)#aflex help time

TIME::clock seconds
  - Returns the current time in the unit of seconds. The function is used
in SMP environment for high-performance processing.

TIME::clock milliseconds
```

```
  - Returns the current time in the unit of milliseconds. The function is
used in SMP environment for high-performance processing. Note: The lowest
resolution of the timer is 4 milliseconds.
```

The following example displays help information for `POLICY::bwlist`:

```
ACOS(config)#aflex help policy::bwlist

POLICY::bwlist id <ip> [<bwlist_name>]
  - Returns the group id of the specified ip address on the black-white
list. If the bwlist_name is not specified, the binded bwlist on the vport
is used.
```

aFleX Script Rename

You do not need to unbind an aFleX script before renaming it. The ACOS device automatically updates the configuration everywhere the renamed script is applied.

aFlex script names can include the following special characters:

- uppercase letters (A-Z)
- lowercase letters (a-z)
- numbers (0-9)
- hyphen (-)
- underscore (_)
- period (.)
- colon (:)

Using the GUI

1. Navigate to **ADC >> aFleX**. The list of configured aFleX scripts appears.
2. Click on the aFleX script name or click Edit in the Actions column to display the configuration page for the script.
3. Edit the name in the Name field.
4. Click **Update**.

The list of aFleX scripts reappears showing the new name. The GUI also automatically updates the aFleX name everywhere the script is used. For example, if the script is already bound to a virtual port, the script's name is automatically updated in the virtual port's configuration. You do not need to manually update the virtual port configuration.

Using the CLI

To rename an aFleX script on the ACOS device, use the following command at the global configuration level of the CLI:

```
ACOS(config)#aflex rename example_old_filename example_new_filename
```

Copy aFleX Script

You can use the CLI to copy an existing aFleX script to a new file with a different name.

From the configuration level of the CLI, use the following command:

```
ACOS(config)#aflex copy example_old_filename example_new_filename
```

NOTE: Scripts that contain syntax errors cannot be copied. The CLI console notifies you if copy failure is due to a syntax error.

Maximum File Size of aFleX Scripts

By default, the maximum file size supported for an aFleX script on an ACOS device is 32 KB. However, this limit can be adjusted to any value between 16 KB and 1024 KB (1 MB).

If Role-Based Access (RBA) is configured on the device, the maximum file size setting applies uniformly to both the shared partition and all private partitions.

To modify the maximum aFleX file size, use the **system resource-usage max-aflex-file-size** command in the global configuration mode. For example, to set the file size limit to 64 KB, use the following:

```
ACOS(config)#system resource-usage max-aflex-file-size 64
```

NOTE: The **exceed-time-limit** counter prevents aFleX scripts from running indefinitely or consuming excessive processing time. If the script exceeds the limit, its execution is aborted, and the counter is incremented. To avoid reaching this limit, it is recommended to optimize your scripts or reduce the number of commands within the script.

Maximum Number of aFleX Scripts

From 4.1.x, ACOS device can have the maximum number of aFleX scripts as the following:

- **Shared**: 1024 (Fixed)
- **Each L3V**: 512 (Fixed)

aFleX Syntax

An aFleX script is a Tcl-like script. Every command call has the following form:

```
command arg1 arg2 arg3 ...
```

The aFleX interpreter takes each word of command call and evaluates it. After evaluation of each word, the first word (command) is considered to be a function name. The function is executed with the rest of the words as arguments.

If a word is surrounded by curly braces { }, this word is unaffected and the substitution is thus not applicable. Inside the braces, there may be spaces and carriage returns. The { } may also be nested.

Local Variable Syntax

In the following example, notice how the line breaks are placed inside the { }.

```
set c example text
if {$c == "Exit"} {
        log "Goobye!"
} else {
        log "Hello!"
```

```
}
```

The first line beginning with `set c` sets the value of the specified local variable. The local variable is only used within the current aFleX script. Replace “example text” with the value you want to set for c; each variable must be set first before it can be called.

The aFleX interpreter sees the remainder of this script as 5 words:

1. '`if`' is the first word. There is nothing to be evaluated.
2. '`$c == "Exit"`' is the second word. Because of the surrounding curly braces, there is no further evaluation on this word.
3. '`log "Goodbye!"`' is the third word. For the same reason as the second word, no further evaluation is needed.
4. '`else`' is the fourth word. There is nothing to be evaluated.
5. '`log "Hello!"`' is the fifth word. No further evaluation is needed.

The first word, '`if`', is taken as the command and this command is executed with the 4 following words as parameters. Later, the condition '`$c == "Exit"`' is evaluated, during the execution of the `if` command.

Use `unset c` to unset the local variable.

Global Variable Syntax

In the following example, notice the difference between using `set/unset` for a local variable and using `table set/delete` for a global variable.

```
when CLIENT_CLOSED {
set client_ip 10.10.10.10
table set active_clients $client_ip 1
   if { [table lookup active_clients $client_ip] != "" }   {
         table incr active_clients $client_ip -1
         if {[table lookup active_clients $client_ip] <= 0 } {
            table delete active_clients $client_ip}
   }
}
```

In the line beginning with set client_ip 10.10.10.10, `client_ip` is a local variable, as previously discussed, which can be used within the current script. The next line beginning with table set active_clients $client_ip 1 sets a global table variable named `active_clients` with a key of `$client_ip` and a value of "1." Replace the name, key, and value with the terminology of your choice. Global table variables can be used by all aFleX scripts.

aFleX Script Components

aFleX scripts consist of the following element types:

- aFleX Operators - Operators are used to compare operands in an expression.
- aFleX Events - When an event is triggered, the policy associated with it will be executed.
- aFleX Commands - Commands are used to query and manipulate data and to direct traffic sent through the ACOS device.

aFleX Context

aFleX scripts support context for specifying either client or server side:

- Each event has a default context of either client-side or server-side.
- Key words: "clientside" or "serverside"
- Only specify the context keywords if you want to change default context.

Example This aFleX script uses the default CLIENT side association to the `REMOTE_ADDR`. Because `CLIENT_ACCEPTED` has a default context of clientside, the `remote_addr` field is automatically assigned to clientside.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::remote_addr] equals 192.168.18.8] } {
        pool www_service_group
    }
}
```

To change the default context of any aFleX script, use the `clientside` or `serverside` key words.

Example This aFleX policy switches the `remote_addr` field to the clientside from the default serverside association with the `SERVER_CONNECTED` event.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [clientside {IP::remote_addr}] equals
192.168.80.81 ] } {
        pool www_service_group
     }
}
```

Tcl Symbols

The Tcl symbols listed in Table 5 have special meanings.

Table 5 : Supported Tcl Symbols in aFlex Policies

Delimeter	Description
`$`	Variable substitution. Example: $argv0 could be replaced by /usr/bin/somescript.tcl
`[ ]`	Subcommand substitution. Example: [pwd] could be replaced by /home/joe
`" "`	Word grouping with substitutions. Example "you are $user" is one word. Substitution still occurs.
`{ }`	Word grouping without substitutions. Example: {you are $user} is one word. $user is not replaced.
`\`	Backslash substitution/escape or statement continuation. By default, a statement ends with the end of the line.
`#`	Comment. This symbol can be used only at the beginning of a statement.
`;`	Statement separator.
`: :`	Namespace path separator for variables or commands. Example: ::foo::bar

For information about standard Tcl syntax, see the following:
http://en.wikibooks.org/wiki/Programming:Tcl

NOTE: Not all Tcl commands and symbols are supported. See Disabled Tcl Commands.

Example aFleX Scripts

Example Pool Selection—This aFleX script uses the `if` command to determine the service group to send traffic based on the file type “html” or “asp”.

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
        pool static_service_group
    } elseif { [HTTP::uri] ends_with ".asp" } {
        pool dynamic_service_group
    }
}
```

Example Node Selection—This aFleX script uses the `node` command to select one specific server to send the traffic to.

```
when HTTP_REQUEST {
        if { [HTTP::uri] ends_with ".gif" } {
                node 192.168.100.10 80
        }
}
```

Example IP Packet Header Query (IP Address)—This example shows that the traffic from client in 192.168.0.0/16 subnet is directed to a special service group called “192_168_service_group”.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.0.0/16] }
{
        pool 192_168_service_group
    } else {
        pool www_service_group
    }
}
```

Example IP Packet Header Query (ToS Level)—This example shows the ToS field being inspected for clientside ToS value of “16”.

```
when CLIENT_ACCEPTED {
    if { [IP::tos] == 16 } {
        pool priority_service_group
    } else {
        pool www_service_group
    }
}
```

Example TCP Query—This aFleX script uses the payload field to check for the words TOP or BOT to properly redirect traffic.

```
when CLIENT_DATA {
    if { [TCP::payload] contains "TOP" } {
        pool top_service_group
    } elseif { [substr[TCP::payload] 50, 3] equals "BOT" } {
        pool bot_service_group
    } else {
        pool www_service_group
    }
}
```

Applying aFleX Scripts To Virtual Ports

These sections describe how to import and bind aFleX scripts.

To use an aFleX policy:

1. Create the aFleX policy. You can create the aFleX policy by typing it into a GUI tab or CLI session, or using a text editor on a PC.
2. Import the aFleX policy onto the ACOS device. You can use the GUI, or the CLI to import the aFleX policy.
3. Bind the aFleX policy to one or more virtual ports. You can bind the aFleX policy to a virtual port using the GUI or CLI.

NOTE: You do not need to unbind an aFleX script before renaming it. The ACOS device automatically updates the configuration wherever the renamed script is used. For more information, see aFleX Script Rename.

The following topics are covered:

aFleX Configuration Prerequisites

- This section describes the prerequisites for aFleX policy configuration.
- For an aFleX policy to take effect, you must bind it to a virtual port on the ACOS device.
- The virtual port must be processing the application type that the Event Declaration in the aFleX policy is triggering on.
- For example, if the aFleX policy includes an event declaration for HTTP_REQUEST, then the policy can only bind to the virtual port that can process HTTP traffic. In other words, the virtual port's service type must be fast-http, http, or https.
- If no aFleX policy is assigned to the virtual port, the ACOS device will continue to redirect traffic to the default server pool (SLB service group) assigned to the virtual port.
- Once an aFleX policy is bound to a virtual port, the policy is triggered whenever the ACOS device encounters the Event Declaration.
- For example, if an aFleX policy includes the event declaration `CLIENT_ACCEPTED`, then the policy is triggered when the ACOS device accepts a client request.
- By default, binding an aFleX policy to a vport marks the vport UP. This functionality can be disabled as follows:

```
ACOS(config)# slb common
ACOS(config-common)# no-auto-up-on-aflex
```

Now, the vport is not automatically marked UP when the aFleX policy is bound, and the vport status will depend on the service group status as usual.

NOTE: For virtual port type fast-HTTP, aFleX commands that change the HTTP header or payload are not supported.

Preloaded aFleX Scripts

Sample aFleX scripts are preloaded onto the ACOS device. This allows you to immediately apply aFleX scripts and build from the provided code.

NOTE: These scripts are intended for educational purposes to assist new users. A10 Networks does not guarantee the sample scripts will work in all contexts and is not liable for damages that result from the misapplication of preloaded aFleX scripts.

See Table 6 for a list of preloaded aFleX scripts.

Table 6 : Preloaded aFleX Scripts

Script	Description
host_ switching	This aFleX example illustrates the use of Tcl associative arrays to implement host switching.
http_ payload_ replace	Collects the HTTP response and then replaces all instances of the pattern “http://” in the payload with “https://”.
logging_ clients	Logs Client/Server IP/Port information for security when using Source NAT.
redirect1	Redirects HTTP requests to an HTTPS URL
redirect2	Uses HTTP::respond to do a redirect with a cookie set.
redirect_ rewrite	Rewrites relative and absolute redirects to absolute HTTPS redirects.

Configure using CLI

The sections mentioned below provide information about adding aFleX scripts in the CLI.

The following topics are covered:

Importing an aFleX Script Using the CLI

1. On a PC that supports TFTP, FTP, SCP, or SFTP, use any text editor to create an aFleX script and save it locally. Use extension ".afx" at the end of the file name. For example, `C:\\aflex\test.afx`
2. On the ACOS device, use the CLI command **`import aflex`** to import the aFleX policy file onto the ACOS device.
3. Use the following command at the configuration level for the virtual port to bind the aFleX script to the virtual port:

   ```
   aflex aflex-name
   ```

   You can specify one script with the command. Repeat the command for each additional script to add.

The scripts will be processed in the order you add them, starting with the first script you add. To re-order the scripts, do either of the following:

- Use the GUI. (See Configure using GUI.)
- In the CLI, use the **`no aflex`** *name* command to remove the scripts from the virtual port, then re-add them in the correct order.

CLI Example

The example mentioned below explains how to import an aFleX policy onto the ACOS device and bind it to a virtual port.

```
when RULE_INIT {
  array set sg_array [list "youtube.com" "sg1" "google.com" "sg2"
"zynga.com" "sg2"]
}

when HTTP_REQUEST {
  set host [HTTP::host]
  if { [info exists $sg_array($host)] } {
    log "host $host -> pool $sg_array($host)"
    pool $sg_array($host)
  }
}
```

1. Log on to the ACOS device through the CLI, and access Global configuration mode.

```
ACOS>enable
Password:
ACOS#config
ACOS(config)#
```

2. Configure nodes (real servers and server ports):

```
ACOS(config)#slb server node100 192.168.9.100
ACOS(config-real server)#port 80 tcp
ACOS(config-real server-node port)#health-check-disable
ACOS(config-real server-node port)#exit
ACOS(config-real server)#exit
ACOS(config)#slb server node101 192.168.9.101
ACOS(config-real server)#port 80 tcp
ACOS(config-real server-node port)#health-check-disable
ACOS(config-real server-node port)#exit
ACOS(config-real server)#exit
ACOS(config)#slb server node102 192.168.9.102
ACOS(config-real server)#port 80 tcp
ACOS(config-real server-node port)#health-check-disable
ACOS(config-real server-node port)#exit
ACOS(config-real server)#exit
ACOS(config)#slb server node103 192.168.9.103
ACOS(config-real server)#port 80 tcp
ACOS(config-real server-node port)#health-check-disable
ACOS(config-real server-node port)#exit
ACOS(config-real server)#exit
ACOS(config)#
```

3. Configure service groups:

```
ACOS(config)#slb service-group http-sg1 tcp
ACOS(config-slb svc group)#member node100 80
ACOS(config-slb svc group-member:80)#exit
ACOS(config-slb svc group)#member node101 80
ACOS(config-slb svc group-member:80)#exit
ACOS(config-slb svc group)#exit
ACOS(config)#slb service-group http-sg2 tcp
```

```
ACOS(config-slb svc group)#member node102 80
ACOS(config-slb svc group-member:80)#exit
ACOS(config-slb svc group)#member node103 80
ACOS(config-slb svc group-member:80)#exit
ACOS(config-slb svc group)#exit
ACOS(config)#
```

4. Use the import command to import the aFleX policy (“test.afx”) onto the ACOS device and rename it “my_aflex”:

```
ACOS(config)#import aflex my_aflex scp://192.168.1.118/aflex/test.afx
User name []?***
Password []?***
Importing ... Done.
ACOS(config)#
```

While importing the aFleX policy, the ACOS device checks for syntax errors. If any syntax errors are found, error messages are displayed. You can modify an aFleX policy and import it again until it passes the syntax check.

5. Use the **show aflex** command to view all aFleX policies on the ACOS device:

```
ACOS(config)#show aflex
Total aFlex number: 7
Max aFlex file size: 32K
Name                    Syntax   Virtual port
-----------------------------------------------
host_switching          Check     No
http_payload_replace    Check     No
http_respond            Check     No
logging_clients         Check     No
my_aflex                Check     No
redirect1               Check     No
redirect2               Check     No
redirect_rewrite        Check     No
```

6. To display the aFleX policy, use the **show aflex** *aflex-name* command:

```
ACOS(config)#show aflex my_aflex
when RULE_INIT {
  array set ::SG_ARRAY [list "youtube.com" "sg1" "google.com" "sg2"
"zynga.com" "sg2"]
```

```
}

when HTTP_REQUEST {
  set host [HTTP::host]
  if { [info exists ::SG_ARRAY($host)] } {
    log "host $host -> pool $::SG_ARRAY($host)"
    pool $::SG_ARRAY($host)
  }
}
```

7. Configure a virtual server and bind the aFleX policy to a virtual port on the virtual server:

```
ACOS(config)#slb virtual-server v30 10.10.8.30
ACOS(config-slb vserver)#port 80 http
ACOS(config-slb vserver-vport)#aflex my_aflex
ACOS(config-slb vserver-vport)#exit
ACOS(config-slb vserver)#exit
ACOS(config)#
```

8. Show the aFleX policy list again to verify that the aFleX policy is now bound to a virtual port:

```
ACOS(config)#show aflex
Total aFlex number: 7
Max aFlex file size: 32K
Name                    Syntax   Virtual port
-----------------------------------------------
host_switching          Check     No
http_payload_replace    Check     No
http_respond            Check     No
logging_clients         Check     No
my_aflex                Check     Bind
redirect1               Check     No
redirect2               Check     No
redirect_rewrite        Check     No
```

9. Show the running-config:

```
ACOS(config)#show running-config
...
```

```
slb server node100 10.10.9.100
   port 80  tcp
       health-check no
slb server node101 10.10.9.101
   port 80  tcp
       health-check no
slb server node102 10.10.9.102
   port 80  tcp
       health-check no
slb server node103 10.10.9.103
   port 80  tcp
       health-check no
!
slb service-group http-sg1 tcp
    member node100 80
    member node101 80
slb service-group http-sg2 tcp
    member node102 80
    member node103 80
!
slb virtual-server v30 10.10.8.30
   port 80  http
      aflex my_aflex
!
...
ACOS(config)#
```

Create an aFleX Script Using the CLI

You can create aFleX policies using the CLI. This feature is especially useful for quickly typing or copy-and-pasting short aFleX scripts. For an example, see Example of Creating an aFleX Script in the CLI.

To configure an aFleX policy using the CLI:

1. Enter the following command at the global configuration level of the CLI:

   ```
   aflex create aflex-name
   ```

   The CLI enters the input mode for the script text.

2. Type or copy-and-paste the script. If you type the script, use the Enter key at the end of each line.
3. To complete the input process, type " . " (period) on a separate line and press Enter.

NOTE:

- You do not need to save the configuration (**`write memory`**) to save the aFleX script. The script is automatically added to a persistent data folder and remains available across reboots.
- Regardless of how an aFleX script is added to the ACOS device, the script does not take effect until you apply it to a virtual port.

Syntax Check

After you finish entering the script text, the CLI performs a syntax check and displays one of the following messages:

- `aFleX aflex-name created; syntax check passed.` – Indicates the syntax is valid.
- `aFleX aflex-name created; syntax check failed.` – Indicates the syntax is not valid. In this case, see Troubleshooting aFleX Syntax Errors.
- `This aFleX already exists.` – Indicates that another aFleX script with the same name is already on the ACOS device.

  The same name can be used in different partitions, but must be unique within a given partition.

Cancelling the aFleX Input Session

To cancel an aFleX script input session before you finish entering the script text, use Ctrl+C. In this case, none of the script is saved.

Example of Creating an aFleX Script in the CLI

The following commands create an aFleX script named "test":

```
ACOS(config)#aflex create test
Type in your aFleX script (type . on a line by itself when done)
when CLIENT_ACCEPTED {
```

```
if {[IP::addr [IP::client_addr] equals 192.168.217.11/24]} {
node 192.168.13.13 80
}
}
aFleX test created; syntax check passed.
```

The following command verifies the script information:

```
ACOS(config)#show aflex test
Name:                    test
Syntax:                  Check
Virtual port:            No
Content:
when CLIENT_ACCEPTED {
if {[IP::addr [IP::client_addr] equals 192.168.217.11/24]} {
node 192.168.13.13 80
}
}
```

The following commands apply the aFleX script to a virtual port:

```
ACOS(config)#slb virtual-server vip1 10.10.10.100
ACOS(config-slb vserver)#port 80 http
ACOS(config-slb vserver-vport)#aflex test
```

Configure using GUI

The following steps and sections describe how to import, create and bind aFleX scripts using the GUI

1. Navigate to **ADC >> aFleX**.

   A list of all configured aFleX scripts appears.

2. Click **Create** to add a new script.

   The Create aFleX page appears.

3. Select Remote or Local:

   - If Remote is selected, you will import a script into the GUI from a remote location. For configuration information, see Create an aFleX Script Using the

GUI.

- If Local is selected, you will input the contents of an aFleX script directly into a field in the GUI. For configuration information, see Import an aFleX Script Using the GUI

4. Bind the aFleX script to a virtual port. For further information about this step, see Bind the aFleX Policy to a Virtual Port.

Create an aFleX Script Using the GUI

On the Create aFleX page:

1. Select the **Local** checkbox in the Local or Remote field.
2. Enter a name for the aFleX policy in the Name field.
3. Enter the aFLeX script into the Definition field.
4. Click **Create** to save the aFleX policy.

NOTE: You edit an aFleX policy by clicking **Edit** in the Actions column next to that aFleX policy's name. You can delete an existing aFleX policy by selecting the checkbox located on the left of its name, then clicking **Delete**.

Figure 2 : ADC > aFleX > Create (Local)

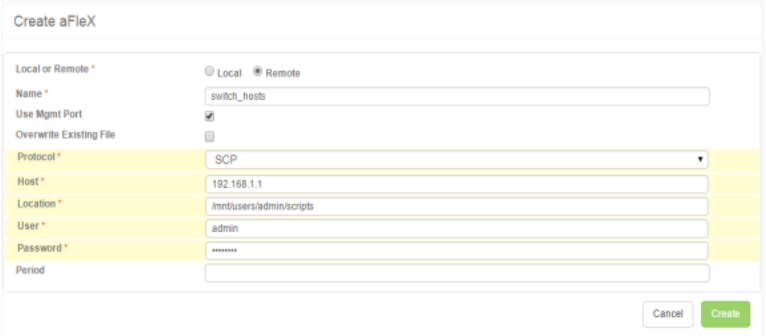
Import an aFleX Script Using the GUI

On the Create aFleX page:

1. Select the **Remote** checkbox in the Local or Remote field.
2. Enter a name for the aFleX policy in the Name field.
3. Specify the transfer protocol that will be used to transfer the file, then provide the necessary credentials to access the remote script.
4. Click **Create**.

NOTE: You edit an aFleX policy by clicking **Edit** in the Actions column next to that aFleX policy's name. You can delete an existing aFleX policy by selecting the checkbox located on the left of its name, then clicking **Delete**.

Figure 3 : ADC > aFleX > Create (Remote)



Bind the aFleX Policy to a Virtual Port

1. Access the configuration settings for the virtual port. You can access them in either of the following ways:
   - Navigate to **ADC >> SLB >> Virtual Servers**, click **Edit** in the Actions column for the virtual server, and then click **Edit** in the Actions column next to the virtual port.
   - Navigate to **ADC >> SLB >> Virtual Services**, and click on the virtual port name.

2. Expand the General Fields section on the page.
3. In the aFleX Scripts field, select one or more aFleX scripts. (See Figure 4.)
4. Click **Update**.

Figure 4 : Add aFleX to Virtual Service or Virtual Port

Troubleshooting aFleX Syntax Errors

After you finish entering the text for an aFleX script, the CLI automatically performs a syntax check. If the check fails, the following message is displayed:

```
aFleX aflex-name created; syntax check failed.
```

In this case, you can fix the script using either CLI or GUI.

The following topics are covered:

Use the CLI to Fix aFleX Syntax Errors

To resolve aFleX syntax errors using the CLI, follow the steps mentioned below:

1. At the global configuration level, use the **`aflex check`** *`name`* command to display syntax error information.
2. Use the **`aflex delete`** *`name`* command to delete the script.
3. Use the **`aflex create`** *`name`* command to re-create the script. (See Create an aFleX Script Using the CLI.)

Use the GUI to Fix aFleX Syntax Errors

Use the GUI to display and edit the script as mentioned below:

1. Navigate to **ADC >> aFleX** to view the aFleX script table.
2. Hover over the aFleX script's name to view the script.
3. Click **Edit** in the Actions column for that script.
4. Change the script text, or re-import the file.
5. Click **Update**.

The aFleX script table reappears. If the script still contains syntax errors, the errors are displayed above the table.

aFleX Operators

aFleX policies use operators to compare operands in an expression.

Available aFleX Operators

The following aFleX operators are supported:

- Logical Operators - used to compare numbers
- Relational Operators - used to compare strings

Other aFleX Components

For information about other script components, see aFleX Script Components.

Logical Operators

Logical operators are used to compare numeric values to one another. They are compatible with all the events, and compatible with any command that has a numeric value (as opposed to a string value).

The following logical operators are supported:

- and
- not
- or

For information about operators, see aFleX Operators.

and

Description Performs a logical “and” comparison between two values.

Syntax

```
<value1> and <value2>
```

Example Use the following example to compare the values for `HTTP::host` and `HTTP::uri`:

```
when HTTP_REQUEST {
    if { ([HTTP::host] equals "www.example.com") and
([HTTP::uri] starts_with "/blog") } {
        pool www_service_group
    } else {
        pool static_service_group
    }
}
```

not

Description Performs a logical “not” on a value.

Syntax

```
not <value>
```

Example Use the following example to see if `HTTP::uri` does not start with a specified string:

```
when HTTP_REQUEST {
    if { not ([HTTP::uri] starts_with "/images") } {
        pool www_service_group
    } else {
        pool static_service_group
    }
}
```

or

Description Performs a logical “or” comparison between two values.

Syntax

```
<value1> or <value2>
```

Example Use the following example to compare two values of `HTTP::uri`:

```
when HTTP_REQUEST {
    if { ([HTTP::uri] starts_with "/images") or ([HTTP::uri]
starts_with "/static") } {
        pool static_service_group
    } else {
        pool www_service_group
    }
}
```

Relational Operators

Relational operators are used to compare strings to one another. They are compatible with all events, and compatible with any command that has a string value (as opposed to a numeric value).

The following relational operators are supported:

- contains
- ends_with

- equals
- matches
- matches_regex
- starts_with

For information about operators, see aFleX Operators.

contains

Description Tests whether one string (string1) contains another string (string2).

Syntax

```
<string1> contains <string2>
```

Example Use the following example test if `HTTP::uri` contains “static”:

```
when HTTP_REQUEST {
    if { [HTTP::uri] contains "static" } {
        pool static_service_group
    } else {
        pool dynamic_service_group
    }
}
```

ends_with

Description Tests whether one string (string1) ends with another string (string2).

Syntax

```
<string1> ends_with <string2>
```

Example Use the following example to test if `HTTP::uri` ends with “.html” or “.asp”:

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
        pool static_service_group
    } elseif { [HTTP::uri] ends_with ".asp" } {
        pool dynamic_service_group
    }
}
```

equals

Description Tests whether one string equals another string.

Syntax

```
<string1> equals <string2>
```

Example Use the following example to test if the domain of `HTTP::host` equals “com”:

```
when HTTP_REQUEST {
    if { [domain [HTTP::host] 1] equals "com" } {
        pool www_service_group
    }
}
```

matches

Description Tests whether one string matches another string.

Syntax

```
<string1> matches <string2>
```

NOTE: The `matches` operator uses the same comparison as the Tcl "string match" command, which functions like a cut-down regular expression.

For the two strings to match, their contents must be identical except that the following special sequences may appear in the pattern:

- `*` – Matches any sequence of characters in string, including a null string.
- `?` – Matches any single character in string.
- `[chars]` – Matches any character in the set given by `chars`. If a sequence of the form x-y appears in `chars`, then any character between x and y, inclusive, will match. When used with `-nocase`, the end points of the range are converted to lower case first. Whereas {[A-z]} matches '_' when matching case-sensitively ('_' falls between the 'Z' and'a'), with `-nocase` this is considered to be like {[A-Za-z]}.

- `\x` – Matches the single character x. This provides a way of avoiding the special interpretation of the characters `*?[]\` in a pattern.

Example Use the following example to test if `HTTP::uri` matches a specified string:

```
when HTTP_REQUEST {
    if { [HTTP::uri] matches {/static[0-9]/*.html} } {
        pool static_service_group
    } else {
        pool dynamic_service_group
    }
}
```

matches_regex

Description Tests whether one string matches a regular expression or another string.

Syntax

```
<string1> matches_regex <regex>
```

The syntax above tests if `<string1>` matches the specified regular expression.

```
<string1> matches_regex <string2>
```

The syntax above tests if `<string2>` is contained within `<string1>`.

Example Use the following example to test if `HTTP::uri` matches a specified regular expression.

```
when HTTP_REQUEST {
    if { [HTTP::uri] matches_regex "^/(static|images)/.*" } {
        pool static_service_group
    } else {
        pool dynamic_service_group
    }
}
```

starts_with

Description Tests whether one string (string1) starts with another string (string2).

Syntax

```
<string1> starts_with <string2>
```

Example Use the following example to test if `HTTP::uri` starts with "/static":

```
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/static" } {
        pool static_service_group
    } else {
        pool dynamic_service_group
    }
}
```

aFleX Events

The following categories of aFleX events are available:

- Global Events
- AAM Events
- Authentication Event
- Database Load-Balancing Events
- Diameter Load-Balancing Events
- DNS Events
- Financial Information eXchange Events
- HTTP Events
- ICAP Events
- IP, TCP, and UDP Events
- MQTT Events
- RAM Caching Events
- SIP Events
- SMTP Events
- SSL Events

Overview

aFleX scripts are event-driven. The ACOS device triggers an aFleX policy based on a specified event. For example, if an aFleX policy is configured to be triggered by the HTTP_REQUEST event, the ACOS device triggers the aFleX policy when an HTTP request is received.

Event declarations are made with the “when” keyword followed by the event name.

Example

```
when CLIENT_ACCEPTED {
        if { [IP::addr [IP::remote_addr] equals 192.168.1.80 ]  } {
                pool example_service_group
        }
}
```

For information about other script components, see aFleX Script Components.

Global Events

This section describes the global events.

For information about aFleX events, see aFleX Events.

The following topics are covered:

RULE_INIT

Description Use this event to immediately set global system variables; when an aFleX script containing a `RULE_INIT` event is added to the virtual server port, the `RULE_INIT` event is immediately triggered and global variables are set.

Within an aFleX policy, the `RULE_INIT` event can initialize a system variable on a global basis for all aFleX policies, or exclusively for that particular aFleX policy.

Syntax

```
when RULE_INIT { <aFleX commands> }
```

The prefix placed before the variable specifies the variable scope. It specifies whether to initialize that variable for all aFleX policies, or only for the current aFleX policy.

Prefix	Scope
`::`	Applies in the same aFlex policy. This variable cannot be set or read by any other aFlex policies. Once a global variable is defined, it cannot be deleted.
`::global::`	Applies to all aFleX policies. This variable can be set or read by all aFleX policies on the ACOS device regardless of partition or CPU.

NOTE: Unbinding an aFleX policy will not remove the variable.

In the current release, it is recommended to avoid using the `unset` command to unset global variables. Doing so may cause a problem. Use `table create/delete` instead.

Usage Valid with the following global variable commands:

- array
- get
- incre
- set

- unset

Example Use the following example to define the number of `HTTP_REQUESTS` using global table variables.

```
when RULE_INIT {
        table set request_count 0 0
        table set ax_request_count 1 1
}
when HTTP_REQUEST {
        table incr request_count 0 1
        table incr ax_request_count 1 2
}
```

LB_FAILED

Description Execute specific aFleX commands when the ACOS device is not able to select a node for the incoming request (for example, if all nodes in the pool are down or all their connection limits have been reached).

When this event is used with aFleX scripts bound to TCP virtual ports, it is triggered by the following conditions:

- The selected server is unreachable (no route host).
- The selected server is non-responsive (fails to respond to a connection request)
- The selected server sent a TCP Reset. In order to enable this trigger, configure **`inband-health-check resel-on-reset`** on a port template attached to the service group or real server port associated with the virtual port.

Syntax

```
when LB_FAILED { <aFleX commands> }
```

Usage Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following class list commands:

- CLASS::exists

- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load-balancing commands:

- LB::server
- LB::status

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside

- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- pool
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Not supported under the ftp-proxy virtual port, such as:

```
ACOS(config)# slb virtual-server vs3 10.10.8.3
ACOS(config-slb vserver)# port 21 ftp-proxy
ACOS(config-slb vserver-vport)# aflex my_aflex
```

Example Use the following example to add a node to the error service group "backup_service_group" when it fails.

```
when LB_FAILED {
    pool backup_service_group
}
```

LB_SELECTED

Description Execute specific aFleX commands when a pool member is selected.

Syntax

```
when LB_SELECTED { <aFleX commands> }
```

Usage Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl

- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing command:

- POLICY::bwlist id

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop

- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- snat
- snatpool
- string map
- substr
- switch
- use
- virtual
- whereis

Not supported under the ftp-proxy virtual port, such as:

```
ACOS(config)# slb virtual-server vs3 10.10.8.3
ACOS(config-slb vserver)# port 21 ftp-proxy
ACOS(config-slb vserver-vport)# aflex my_aflex
```

Example Use the following example to add a pool member to a source NAT pool when the member is selected.

```
when LB_SELECTED {
    if { [IP::addr [IP::remote_addr] equals "192.168.8.8"] } {
        snatpool snat-internal
    }
}
```

CONTROL_TIMER_ONE_SEC

Description Execute specific aFleX commands every second, typically used for periodic tasks such as health checks, time-based logging, or other recurring actions.

Syntax

```
when CONTROL_TIMER_ONE_SEC { <aFleX commands> }
```

Example Use this example to log the current hit count stored in a table every second.

```
when CONTROL_TIMER_ONE_SEC {
    set count [table lookup my_table hits]
    log "Hits so far: $count"
}
```

PACKET_RECEIVED

Description Execute specific aFleX commands when a packet is received from the client, before it is fully processed by higher-layer protocols. This event is used for packet-level inspection, logging, filtering, or redirection.

Syntax

```
when PACKET_RECEIVED{ <aFleX commands> }
```

Example Use the following example to log incoming packet data:

```
when PACKET_RECEIVED {
    set payload [TCP::payload]
    log "Received raw payload: $payload"
}
```

AAM Events

The following Authentication Authorization Management (AAM) events are available:

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_AUTHORIZATION_INIT
- AAM_RELAY_INIT
- For information about aFleX events, see aFleX Events.
- For information about AAM commands, see AAM Commands.

NOTE: aFleX scripts containing AAM events are only valid on HTTP and HTTPS virtual ports. Also, AAM events are not triggered for OCSP configurations.

AAM_AUTHENTICATION_INIT

Description Execute specific aFleX scripts during preparation before AAM authentication.

Syntax

```
when AAM_AUTHENTICATION_INIT { <aFleX commands> }
```

NOTE: This event is not triggered for OCSP configurations.

Usage Valid for the following AAM commands:

- AAM::attribute_collection
- AAM::authentication
- AAM::authorization
- AAM::client
- AAM::relay
- AAM::session

Valid with the following category commands:

- CATEGORY::lookup

Valid for the following global commands:

- active_members
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons

- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Valid with the following HTTP commands:

- HTTP::password
- HTTP::username

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following URL commands:

- URL::reputation

Example Use the following example to append a different prefix to the username for authentication and relay. The ACOS device will use the username `AUTH_$name` for authentication, `RELAY_$name` for relay, and `$name` for authorization.

```
when AAM_AUTHENTICATION_INIT {
        set name [AAM::client get username]
        AAM::authentication set username "AUTH_$name"
        AAM::relay set username "RELAY_$name"
        }
```

For additional examples, see Example 7: Getting a constructed JWT from a Session.

AAM_AUTHORIZATION_INIT

Description Execute specific aFleX scripts in preparation for AAM authentication and relay.

Syntax

```
when AAM_AUTHENTICATION_INIT { <aFleX commands> }
```

NOTE: This event is not triggered for OCSP configurations.

Usage Valid for the following AAM command:

- AAM::session

Valid for the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event

- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Valid with the following HTTP commands:

- HTTP::password
- HTTP::username

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following URL command:

- URL::reputation

Example For additional examples, see Example 7: Getting a constructed JWT from a Session.

AAM_AUTHORIZATION_CHECK

Description Execute specific aFleX commands for AAM authorization.

Syntax

```
when AAM_AUTHORIZATION_CHECK { <aFleX commands> }
```

NOTE: This event is not triggered for OCSP configurations.

Usage Valid for the following AAM commands:

- AAM::attribute
- AAM::attribute_collection
- AAM::authentication
- AAM::authorization
- AAM::client
- AAM::session

Valid with the following category command:

- CATEGORY::lookup

Valid for the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event

- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Valid with the following HTTP commands:

- HTTP::password
- HTTP::username

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following URL command:

- URL::reputation

Example See Example 7: Getting a constructed JWT from a Session.

AAM_RELAY_INIT

Description Execute specific aFleX scripts during preparation before AAM relay.

Syntax

```
when AAM_RELAY_INIT { <aFleX commands> }
```

NOTE: This event is not triggered for OCSP configurations.

Usage Valid for the following AAM commands:

- AAM::attribute
- AAM::authentication
- AAM::authorization
- AAM::client
- AAM::relay
- AAM::session

Valid with the following category command:

- CATEGORY::lookup

Valid for the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr

- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Valid with the following HTTP commands:

- HTTP::password
- HTTP::username

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following URL command:

- URL::reputation

Example Use the following example to append a different prefix to the username for relay. The ACOS device will append `AUTH_` to the username, and set this new username to `$relay_name` for relay.

```
when AAM_RELAY_INIT {
   set relay_name "AUTH_"
   append relay_name [AAM::relay get username]
   AAM::relay set username $relay_name
}
```

Authentication Event

The following Authentication event is available:

- AUTH_RESULT

For information about aFleX commands, see aFleX Commands.

AUTH_RESULT

Description Execute specific aFleX commands when an authentication result is received.

Syntax

```
when AUTH_RESULT { <aFleX commands> }
```

Example Use the following example to handle authentication failure when the request is not explicitly released:

```
when AUTH_RESULT {
    HTTP::respond 403 content "Authentication Failed - Access
Denied"
}
```

Database Load-Balancing Events

The following database load-balancing (DBLB) events are available:

- DB_COMMAND
- DB_QUERY
- DB_RESPONSE

For information about aFleX events, see aFleX Events.

For information about DBLB commands, see Database Load-Balancing Commands.

DB_COMMAND

Description Execute specific aFleX commands when an SQL command is sent by the client.

Syntax

```
when DB_COMMAND { <aFleX commands> }
```

Usage Valid for the following database load balancing commands:

- DB::command

Valid for the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist

- reject
- return
- serverside
- set encode
- sha1
- snat
- string map
- substr
- switch
- use
- virtual

Example Use the following example to create a log entry to “mssql_service_group” whenever an SQL command is sent by the client.

```
when DB_COMMAND {
    log "DB Command: [DB::command]"
    pool mssql_service_group
}
```

DB_QUERY

Description Execute specific aFleX commands when a full SQL query is received from the client.

Syntax

```
when DB_QUERY { <aFleX commands> }
```

Usage Valid for the following database load balancing commands:

- DB::query

Valid for the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage

- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- snat
- string map
- substr
- switch
- use
- virtual

Example Use the following example to separate database read queries from write or other commands. The `service-group sg-mysql-write` includes only the master MySQL server, where all write operations and other DB

commands are executed. In contrast, `service-group sg-mysql-read` contains all MySQL servers used as a pool to handle read queries.

```
when DB_QUERY {
        set ret [string tolower [ DB::query ]]
        log "aflex script got query: $ret"
        if { ($ret starts_with "insert") or ($ret starts_with
"update") or ($ret starts_with "delete") } {
                    log "aflex got a write command: $ret"

                    pool sg-mysql-write
            } else {
                    log "aflex got a read command: $ret"
                    pool sg-mysql-read       }

            }
when DB_COMMAND {
      set ret [ DB::command ]
      log "aflex script got command number: $ret"
      pool sg-mysql-write
}
```

DB_RESPONSE

Description Execute specific aFleX commands when the server sends a complete response to the SQL query initiated by the client.

Syntax

```
when DB_RESPONSE { <aFleX commands> }
```

Example Use the following example to log DB command after server response:

```
when DB_RESPONSE {
      set ret [ DB::command ]
      log "Command type received: $ret"
}
```

Diameter Load-Balancing Events

The following diameter load-balancing events are available:

- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND

For information about aFleX events, see aFleX Events.

For information about diameter load-balancing commands, see Diameter Load-Balancing Commands.

DIAMETER_ANSWER

Description Execute specific aFleX commands when a complete Diameter answer message is fully parsed.

Syntax

```
when DIAMETER_ANSWER { <aFleX commands> }
```

Usage Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members

- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to create a log entry whenever a Diameter answer message is fully parsed.

```
when DIAMETER_ANSWER {
        log "DIAMETER::cmd_code = [DIAMETER::cmd_code]"
}
```

DIAMETER_ANSWER_SEND

Description Execute specific aFleX commands immediately before a Diameter answer is sent.

Syntax

```
when DIAMETER_ANSWER_SEND { <aFleX commands> }
```

Usage Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use

- virtual

Example Use the following example to create a log entry immediately before a Diameter answer is sent.

```
when DIAMETER_ANSWER_SEND {
        log "DIAMETER::cmd_code = [DIAMETER::cmd_code]"
}
```

DIAMETER_REQUEST

Description Execute specific aFleX commands when a complete Diameter request message is fully parsed.

Syntax

```
when DIAMETER_REQUEST { <aFleX commands> }
```

Usage Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl

- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to create a log entry whenever a Diameter request message is fully parsed.

```
when DIAMETER_REQUEST {
        log "DIAMETER::cmd_code = [DIAMETER::cmd_code]"
}
```

DIAMETER_REQUEST_SEND

Description Execute specific aFleX commands immediately before a Diameter request is sent.

Syntax

```
when DIAMETER_REQUEST_SEND { <aFleX commands> }
```

Usage Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode

- sha1
- string map
- substr
- switch
- use
- virtual

Example To check the Origin-Realm value inside the incoming Diameter Request, send a Diameter Response 3003 (Intended Realm is not recognized) back to client if it is not "test.com".

```
when DIAMETER_REQUEST {
  set dropflag 0
  if { !\([DIAMETER::avp [DIAMETER::avp get_ids 296] value]
equals "test.com")}{
    log " flag if Diameter AVP Origin-Realm is NOT test.com "
    set dropflag 1
  }
}
when DIAMETER_ANSWER_SEND {
if { $dropflag } {
    log "Remove server response code and return code 3003 to
client "
    DIAMETER::avp [DIAMETER::avp get_ids 268] delete
    DIAMETER::avp insert 268 3003 -M-
  }
when DIAMETER_REQUEST_SEND {
        log "DIAMETER::cmd_code = [DIAMETER::cmd_code]"
}
```

DNS Events

The following DNS events are available:

- DNS_REQUEST
- DNS_RESPONSE

For information about aFleX events, see aFleX Events.

For information about DNS commands, see DNS Commands.

DNS_REQUEST

Description Execute specific aFleX commands when DNS request packets arrive.

Syntax

```
when DNS_REQUEST { <aFleX commands> }
```

Usage Valid with the following category command:

- CATEGORY::lookup

Usage Valid with the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following DNS commands:

- DNS::additional
- DNS::answer
- DNS::authority
- DNS::cache
- DNS::class
- DNS::header
- DNS::is_dnssec
- DNS::len
- DNS::name
- DNS::query
- DNS::question
- DNS::rdata
- DNS::return
- DNS::rr
- DNS::ttl
- DNS::type

Valid with the following limit ID commands:

- IP::category
- IP::reputation

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl

- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following command to log the length of DNS queries received:

```
when DNS_REQUEST {
    log "DNS Len: [DNS::len]"
}
```

Use the following commands to perform a category lookup based on the queried domain name:

```
when DNS_REQUEST {
log "Received DNS request for: [DNS::question name]"
set query_name [DNS::question name]

set cat [CATEGORY::lookup $query_name]
foreach cat $cats {
log  "HTTP request: num: category: $cat"
if {$cat == "search-engines"} {
   log "match"
}
}
DNS::return
}
```

DNS_RESPONSE

Description Execute specific aFleX commands when DNS reply packets arrive.

Syntax

```
when DNS_RESPONSE { <aFleX commands> }
```

Usage Valid with the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following DNS commands:

- DNS::additional
- DNS::answer
- DNS::authority
- DNS::cache
- DNS::class
- DNS::header
- DNS::is_dnssec
- DNS::len
- DNS::name
- DNS::query
- DNS::question
- DNS::rdata
- DNS::return
- DNS::rr
- DNS::ttl
- DNS::type
-

Valid with the following limit ID commands:

- IP::category

- IP::reputation

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs

- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to log the length of DNS reply packets received.

```
when DNS_RESPONSE {
    log "DNS Len: [DNS::len]"
}
```

Financial Information eXchange Events

The following Financial Information eXchange (FIX) events are available:

- FIX_REQUEST
- FIX_RESPONSE

For information about aFleX events, see aFleX Events.

For information about FIX commands, see Financial Information eXchange Commands.

FIX_REQUEST

Description Execute specific aFleX commands when a FIX request is received.

Syntax

```
when FIX_REQUEST { <aFleX commands> }
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Usage Valid with the following FIX commands:

- FIX::begin_string
- FIX::body_length
- FIX::msg_seq_num
- FIX::msg_type
- FIX::sender_compid
- FIX::sending_time
- FIX::target_compid

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if

- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to group traffic in `fix_client_service_group` whenever a FIX request is received.

```
when FIX_REQUEST {
    if { [FIX::sender_compid] eq "CLIENT1" } {
        pool fix_client_service_group
    }
}
```

FIX_RESPONSE

Description Execute specific aFleX commands when a FIX response is received.

Syntax

```
when FIX_RESPONSE { <aFleX commands> }
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Usage

Valid with the following FIX commands:

- FIX::begin_string
- FIX::body_length
- FIX::msg_seq_num
- FIX::msg_type
- FIX::sender_compid
- FIX::sending_time
- FIX::target_compid

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist

- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to log FIX information as specified when a FIX response is received.

```
when FIX_RESPONSE {
    log "[FIX::sender_compid] -> [FIX::target_compid]"
}
```

HTTP Events

The following HTTP events are available:

- HTTP_RESPONSE_DATA
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE
- HTTP_REQUEST_SEND
- HTTP_REQUEST_DATA
- HTTP_REQUEST

For information about aFleX events, see aFleX Events.

For information about HTTP commands, see HTTP Commands.

HTTP_REQUEST

Description Execute specific aFleX commands when a complete client request header (method, URI, version, and all headers, not including the body) is parsed.

For information about parsing WebDAV messages, see Parsing WebDAV Messages in the "Usage" section below.

Syntax

```
when HTTP_REQUEST { <aFleX commands> }
```

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::bypass
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following RAM caching commands:

- CACHE::age
- CACHE::disable
- CACHE::enable
- CACHE::expire
- CACHE::headers
- CACHE::hits

valid with the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following category commands:

- CATEGORY::lookup

Valid with the following HTTP commands:

- HTTP::close
- HTTP::collect
- HTTP::cookie
- HTTP::header
- HTTP::host

- HTTP::is_keepalive
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::password
- HTTP::payload
- HTTP::query
- HTTP::redirect
- HTTP::release
- HTTP::request
- HTTP::request_num
- HTTP::respond
- HTTP::retry
- HTTP::status
- HTTP::stream
- HTTP::uri
- HTTP::username
- HTTP::version

Valid with the following IP commands:

- IP::addr
- IP::category
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::reputation
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load-balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based load balancing command:

- POLICY::bwlist id
- POLICY::source_rule

Valid with the following DNS resolution command:

- RESOLVE::lookup

Valid with the following SSL commands:

- SSL::cert
- SSL::cipher
- SSL::renegotiate
- SSL::session invalidate
- SSL::sessionid
- SSL::verify_result

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following statistics commands:

- STATS::clear

Valid with the following URI commands:

- URI::basename
- URI::decode
- URI::encode
- URI::path
- URI::query

Valid with the following URL commands:

- URL::reputation

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode

- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- lwnode
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- pool
- reject
- return
- serverside
- session
- set encode
- sha1
- snat
- snatpool
- string map
- substr

- switch
- use
- virtual
- whereis

Parsing WebDAV Messages

aFleX can parse certain WebDAV methods. During an HTTP_REQUEST event, the specified WebDAV methods have been used, the body of the code will be executed. WebDAV methods only detected as events; they cannot be implemented at this time. The following WebDAV methods are supported:

- COPY - creates a duplicate of a resource.
- DELETE - deletes a resource.
- LOCK - locks a resource to prevent users from editing simultaneously.
- MKCOL - creates a new collection in which resources can be stored.
- MOVE - moves a resource to a new location.
- PROPFIND - retrieves the properties of a resource.
- PROPPATCH - sets the properties of a resource.
- UNLOCK- unlocks a resource, negating the lock command

Example Use the following example to redirect the client to HTTPS if a client request URI contains the string "secure":

```
when HTTP_REQUEST {
    if { [HTTP::uri] contains "secure" } {
        HTTP::redirect "https://[HTTP::host][HTTP::uri]"
    }
}
```

Example Use this example to group traffic based on the WebDAV method in the HTTP request header.

```
when HTTP_REQUEST {
    if { not ([HTTP::method] equals "PROPFIND") } {
        if { [IP::addr [IP::client_addr] equals
192.168.1.0/24] } {
```

```
            pool davwriters_service_group
        }
    } else {
        pool davreaders_service_group
    }
}
```

HTTP_REQUEST_DATA

Description Execute specific aFleX commands when an `HTTP::collect` command is finished processing.

Syntax

```
when HTTP_REQUEST_DATA { <aFleX commands> }
```

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable

- COMPRESS::enable
- COMPRESS::gzip

Valid with the following category commands:

- CATEGORY::lookup

Valid with the following HTTP commands:

- HTTP::close
- HTTP::collect
- HTTP::cookie
- HTTP::header
- HTTP::host
- HTTP::is_keepalive
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::password
- HTTP::payload
- HTTP::query
- HTTP::redirect
- HTTP::release
- HTTP::request
- HTTP::request_num
- HTTP::respond
- HTTP::retry
- HTTP::status
- HTTP::stream
- HTTP::uri
- HTTP::username
- HTTP::version

Valid with the following IP commands:

- IP::addr

- IP::category
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::reputation
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load-balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id
- POLICY::source_rule

Valid with the following DNS resolution command:

- RESOLVE::lookup

Valid with the following SSL commands:

- SSL::cert
- SSL::cipher

- SSL::renegotiate
- SSL::session invalidate
- SSL::sessionid
- SSL::verify_result

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following URI commands:

- URI::basename
- URI::decode
- URI::encode
- URI::path
- URI::query

Valid with the following URL commands:

- URL::reputation

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject

- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- lwnode
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- pool

- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to record a persist variable after data is collected by the `HTTP::collect` command, then log the recorded variable.

```
when HTTP_REQUEST_DATA {
    set rpc_var [findstr [HTTP::payload] "Authorization:" 14
20]
    persist uie $rpc_var
    log "Persist UIE: $rpc_var"
    HTTP::release
}
```

HTTP_REQUEST_SEND

Description Execute specific aFleX commands immediately before a request is sent to a server. This is a server-side event.

Syntax

```
when HTTP_REQUEST_SEND { <aFleX commands> }
```

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following category commands:

- CATEGORY::lookup

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following IP commands:

- IP::addr
- IP::category
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::reputation
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::source_rule

Valid with the following SSL commands:

- SSL::cert
- SSL::cipher
- SSL::session invalidate
- SSL::sessionid
- SSL::verify_result

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number

- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist

- pool
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to begin collecting TCP data immediately before an HTTP request is sent to a server.

```
when HTTP_REQUEST_SEND {
    HTTP::collect
}
```

HTTP_RESPONSE

Description Execute specific aFleX commands when all of the response status and header lines from the server response are parsed.

Syntax

```
when HTTP_RESPONSE { <aFleX commands> }
```

NOTE: `HTTP_RESPONSE` is specific to a SERVER response passing through the load balancer, and is not triggered for locally-generated responses.

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following category commands:

- CATEGORY::lookup

Valid with the following RAM caching commands:

- CACHE::age
- CACHE::disable
- CACHE::enable
- CACHE::expire
- CACHE::headers
- CACHE::hits

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following HTTP commands:

- HTTP::close
- HTTP::collect
- HTTP::cookie
- HTTP::header

- HTTP::host
- HTTP::is_keepalive
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::query
- HTTP::redirect
- HTTP::release
- HTTP::request
- HTTP::request_num
- HTTP::respond
- HTTP::retry
- HTTP::status
- HTTP::stream
- HTTP::version

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit

- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id
- POLICY::source_rule

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following URI commands:

- URI::basename
- URI::decode
- URI::encode
- URI::path
- URI::query

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside

- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- session
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use this example to redirect an HTTP request to "http://backup.exampledomain.com" whenever an HTTP response is fully parsed and a 404 error has occurred:

```
when HTTP_RESPONSE {
    if { [HTTP::status] == 404 } {
        HTTP::redirect "http://backup.exampledomain.com"
    }
}
```

HTTP_RESPONSE_CONTINUE

Description Execute specific aFleX commands whenever the system receives a 100 Continue response from the server.

Syntax

```
when HTTP_RESPONSE_CONTINUE { <aFleX commands> }
```

Example Use the following example to create a log entry whenever a "100-Continue" response is received from the server and the HTTP version is other than 1.1:

```
when HTTP_RESPONSE_CONTINUE {
    if { [HTTP::version] != 1.1 } {
        log "Bad server: sent 100-Continue to non-1.1 client."
    }
}
```

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following category commands:

- CATEGORY::lookup

Valid wtih the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following IP command:

- IP::server_addr

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::source_rule

Valid with the following TCP commands:

- TCP::client_port
- TCP::close

- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1

- string map
- substr
- switch
- use
- virtual
- whereis

HTTP_RESPONSE_DATA

Description Execute specific aFleX commands when an `HTTP::collect` command finishes processing on the server side of a connection.

Syntax

```
when HTTP_RESPONSE_DATA { <aFleX commands> }
```

NOTE: This event is also triggered if the server closes the connection before the `HTTP:collect` command finishes processing.

Example Use the following example to replace the string "https://site1.exampledomain.com" with "https://site2.exampledomain.com" after `HTTP:: collect` has finished processing. The new string is saved into the HTTP payload and sent to the client.

```
when HTTP_RESPONSE {
if { ([HTTP::status] == 200) and ([HTTP::header "Content-
Type"] contains "text") } {
    if { [HTTP::header exists Content-Length] } {
        HTTP::collect [HTTP::header Content-Length]
    } else {
        HTTP::collect
    }
}
}
when HTTP_RESPONSE_DATA {
    regsub -all "http://site1.exampledomain.com/"
[HTTP::payload] "https://site2.exampledomain.com" newpayload
    HTTP::payload replace 0 [HTTP::payload length] $newpayload
    HTTP::release
}
```

Usage Valid with the following Application Access Management (AAM) commands:

- AAM::attribute
- AAM::client
- AAM::session

Valid with the following Advanced Encryption Standard (AES) commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following category commands:

- CATEGORY::lookup

Valid with the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following HTTP commands:

- HTTP::close
- HTTP::collect
- HTTP::header
- HTTP::host
- HTTP::is_keepalive
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::query
- HTTP::redirect

- HTTP::release
- HTTP::request
- HTTP::request_num
- HTTP::respond
- HTTP::retry
- HTTP::status
- HTTP::stream
- HTTP::version

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load-balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id
- POLICY::source_rule

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::client_port
- TCP::close
- TCP::local_port
- TCP::mss
- TCP::rtt
- TCP::server_port

Valid with the following URI commands:

- URI::basename
- URI::decode
- URI::encode
- URI::path
- URI::query

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr

- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

ICAP Events

The following Internet Content Adaptation Protocol (ICAP) events are available:

- ICAP_REQUEST
- ICAP_RESPONSE

For information about aFleX events, see aFleX Events.

ICAP_REQUEST

Description Triggered when ICAP command is created but before being sent to ICAP server.

Syntax

```
when ICAP_REQUEST { <aFleX Command> }
```

Usage Valid with the following ICAP commands:

- ICAP::header add
- ICAP::header remove
- ICAP::header replace
- ICAP::header replace-all
- ICAP::method
- ICAP::uri

Example Use the following command to define an ICAP URI to route traffic through an ICAP server when an ICAP_REQUEST event is triggered:

```
when ICAP_REQUEST {
    ICAP::uri icap://A10icap:1344/echo
}
```

ICAP_RESPONSE

Description Triggered after ICAP response has been processed but before result is sent to the virtual server.

Syntax

```
when ICAP_RESPONSE { <aFlex Command> }
```

Usage Valid with the following ICAP commands:

- ICAP::header values
- ICAP::status
- HTTP::close
- HTTP::cookie
- HTTP::header

- HTTP::host
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::redirect
- HTTP::request_num
- HTTP::respond
- HTTP::status
- HTTP::uri

Example Use the following command to log the ICAP response status and the value of the 'ISTag' header when an ICAP_RESPONSE event is triggered:

```
when ICAP_RESPONSE {
       log "ICAP response code is [ICAP::status]"
       log "ISTag header value is [ICAP::header values ISTag]"
       }
```

IP, TCP, and UDP Events

The following events related to IP, TCP and UDP traffic are available:

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

For information about aFleX events, see aFleX Events.

See the following sections for information about commands related to these events:

- IP Commands
- TCP Commands
- UDP Commands

CLIENT_ACCEPTED

Description Execute specific aFleX commands when a client establishes a connection with the ACOS device.

For TCP, the `CLIENT_ACCEPTED` event is triggered as follows:

- For L4 without syn-cookie, it is triggered on the first packet.
- For L4 with syn-cookie and L7, it is triggered when a TCP handshake is completed.

Syntax

```
when CLIENT_ACCEPTED { <aFleX commands> }
```

NOTE: For UDP (and only UDP), the `CLIENT_ACCEPTED` event is triggered on the first UDP packet received.

Usage Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class-list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following diameter load-balancing commands:

- DIAMETER::app_id

- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following IP commands:

- IP::addr
- IP::category
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::reputation
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop

- LINK::vlan_id

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following SSL commands:

- SSL::disable
- SSL::enable
- SSL::mode
- SSL::template

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::collect
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members

- b64decode
- b64encode
- clientside
- cpu usage
- discard
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- lwnode
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- pool
- reject
- return
- serverside
- session
- set encode
- sha1
- sha256
- snat

- snatpool
- string map
- substr
- switch
- use
- virtual
- when
- whereis

Example Use the following example to log the time whenever a connection is established:

```
when CLIENT_ACCEPTED {
    log "Client [IP::client_addr] connected at [clock format
[TIME::clock seconds] -format {%T}]"
}
```

Example Use the following example to discard a specific IP address when a connection is established.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [client_addr] equals 192.168.1.2/24] } {
        log "Discard connection from [IP::client_addr]"
        discard
    }
}
```

CLIENT_CLOSED

Description Execute specific aFleX commands at the end of any client connection, regardless of protocol.

Syntax

```
when CLIENT_CLOSED { <aFleX commands> }
```

Usage Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return

- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to decrement the client IP counter by 1 each time a client connection is closed. If all connections from the specified client IP are closed, the counter is deleted.

```
when CLIENT_CLOSED {
set $client_ip 10.10.10.10
table set active_clients $client_ip 1
   if { [table lookup active_clients $client_ip] != "" }   {
         table incr active_clients $client_ip -1
         if {[table lookup active_clients $client_ip] <= 0 } {
            table delete active_clients $client_ip}
   }
}
```

CLIENT_DATA

Description Execute specific aFleX commands when new data is received from the client while the connection is in a collect state.

Syntax

```
when CLIENT_DATA { <aFleX commands> }
```

NOTE: For UDP, the `CLIENT_DATA` event is automatically triggered for each UDP packet received. IP fragmentation of a UDP packet is not supported for the CLIENT_DATA event.

Usage Valid with the following AES commands:

- AES::decrypt

- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following IP command:

- IP::server_addr

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool

- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following RADIUS commands:

- RADIUS::avp
- RADIUS::code
- RADIUS::id
- RADIUS::length

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::collect
- TCP::local_port
- TCP::mss
- TCP::notify
- TCP::offset
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::release
- TCP::remote_port

- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- lwnode
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- pool

- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to select the service group “top_dns_service_group” when a new client DNS request contains “TOP”:

```
when CLIENT_DATA {
        if { [UDP::payload 50] contains "TOP" } {
                pool top_dns_service_group
        }
}
```

Example Use the following example to select service group “one_dns_service_group” when a new client DNS request contains “one”, or select service group “two_dns_service_group” when a new client DNS request contains “two”. In either case, a log entry is also created.

```
when CLIENT_DATA {
        log "UDP::payload 12 12 = [UDP::payload 12 12]"
        if { [UDP::payload 12 12] contains "one" } {
                pool one_dns_service_group
                log " service group one_dns_service_group was selected"
        } elseif { [UDP::payload 12 12] contains "two" } {
                pool two_dns_service_group
                log " service group two_dns_service_group was selected"
        }
}
```

SERVER_CLOSED

Description Execute specific aFleX commands when the server-side connection closes.

Syntax

```
when SERVER_CLOSED { <aFleX commands> }
```

Valid Events Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following diameter load balancing commands:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

Valid with the following IP commands:

- IP::addr
- IP::client_addr

- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::local_port

- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5

- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to generate a log message containing the IP address of the server whenever a server-side connection is closed:

```
when SERVER_CLOSED {
    log "Server [IP::server_addr] has closed the connection"
}
```

SERVER_CONNECTED

Description Execute specific aFleX commands when a connection is established with the server.

Syntax

```
when SERVER_CONNECTED { <aFleX commands> }
```

Usage Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following load balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit

- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::collect
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

- whereis

Example Use this example to create variables that include the IP addresses and TCP ports of the client and the server when a server connection is established:

```
when CLIENT_ACCEPTED {
    set vip "[IP::local_addr]:[TCP::local_port]"
}
when SERVER_CONNECTED {
    set client "[IP::client_addr]:[TCP::client_port]"
    set node "[IP::server_addr]:[TCP::server_port]"
}
when CLIENT_CLOSED {
    log "Client $client -> VIP: $vip -> Node: $node"
}
```

SERVER_DATA

Description Execute specific aFleX commands when new data is received from the server while the connection is in a hold state.

Syntax

```
when SERVER_DATA { <aFleX commands> }
```

NOTE: For UDP, the SERVER_DATA event is triggered for every packet. For TCP, you need to issue a TCP::collect.

Usage Valid with the following AES commands:

- AES::decrypt
- AES::encrypt

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following compression commands:

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

Valid with the following load-balancing commands:

- LB::server
- LB::status

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following link commands:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

Valid with the following IP command:

- IP::server_addr

Valid with the following policy-based server load balancing (PBSLB) command:

- POLICY::bwlist id

Valid with the following RADIUS commands:

- RADIUS::avp
- RADIUS::code
- RADIUS::id
- RADIUS::length

valid with the following statistics commands:

- STATS::clear

Valid with the following TCP commands:

- TCP::close
- TCP::collect
- TCP::local_port
- TCP::mss
- TCP::notify
- TCP::offset
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::release
- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop

- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual
- whereis

Example Use the following example to define the variable `payload` whenever new data is received from the server while the connection is in a hold state:

```
when SERVER_DATA {
    log "TCP Payload: [TCP::payload]"
}
```

MQTT Events

The following MQTT events are available:

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA
- MQTT_PUBLISH
- MQTT_SUBSCRIBE

For information about aFleX events, see aFleX Events.

For information about MQTT commands, see MQTT Commands.

MQTT_CLIENT_MESSAGE

Description Executes the specific aFleX scripts when a client sends an MQTT message.

Syntax

```
when MQTT_CLIENT_MESSAGE { <aFleX commands> }
```

Example Use the following example to log a client id:

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::client_id]"

}
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::collect
- MQTT::drop
- MQTT::dup_flag

- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::retain_flag
- MQTT::return_code
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

MQTT_SERVER_MESSAGE_DATA

Description Triggered only when MQTT::collect finishes collecting under MQTT_SERVER_MESSAGE

Syntax

```
MQTT_SERVER_MESSAGE_DATA { <aFleX commands> }
```

Example Use the following commands to collect and log MQTT message data when a PUBLISH message is received from the server:

```
when MQTT_SERVER_MESSAGE {
    MQTT::collect
}

when MQTT_SERVER_MESSAGE_DATA {
    if { [MQTT::type] equals 8} {
```

```
        log "payload in PUBLISH is [MQTT::payload]"
    }
}
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::return_code
- MQTT::return_code_list
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

MQTT_SERVER_MESSAGE

Description Executes the specific aFleX scripts when a server sends an MQTT message.

Syntax

```
when MQTT_SERVER_MESSAGE {<aFleX commands>}
```

Example Use the following example to log a payload:

```
when MQTT_SERVER_MESSAGE {
        log "[MQTT::payload]"
                }
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::collect
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::return_code
- MQTT::return_code_list
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

MQTT_CLIENT_MESSAGE_DATA

Description Triggered only when MQTT::collect finishes collecting under MQTT_CLIENT_MESSAGE

Syntax

```
when MQTT_CLIENT_MESSAGE { <aFleX commands> }
```

Example Use the following commands to collect and log MQTT message data when a PUBLISH message is received from the client:

```
when MQTT_CLIENT_MESSAGE {
    MQTT::collect
}

when MQTT_CLIENT_MESSAGE_DATA {
    if { [MQTT::type] equals 8} {
        log "payload in PUBLISH is [MQTT::payload]"
    }
}
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond

- MQTT::retain_flag
- MQTT::return_code
- MQTT::return_code_list
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

MQTT_PUBLISH

Description Executes the specific aFleX scripts when a broker publishes an MQTT PUBLISH message.

Syntax

```
when MQTT_PUBLISH { <aFleX commands> }
```

Example Use the following example to log a topic:

```
when MQTT_PUBLISH {
        log "[MQTT::topic]"
}
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::collect
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name

- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::retain_flag
- MQTT::return_code
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

MQTT_SUBSCRIBE

Description Executes the specific aFleX scripts when a client subscribes to an MQTT SUBSCRIBE message.

Syntax

```
when MQTT_SUBSCRIBE {<aFleX commands>}
```

Example Use the following example to log a topic:

```
when MQTT_SUBSCRIBE {
       log "[MQTT::topic]"
}
```

Usage Valid for all MQTT commands:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::collect
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password

- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::return_code
- MQTT::return_code_list
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type
- MQTT::username
- MQTT::will

RAM Caching Events

The following RAM caching events are available:

- CACHE_REQUEST
- CACHE_RESPONSE

NOTE: These commands are supported on HTTP traffic (the original proxy), but not supported on HTTP2 traffic (or the new proxy).

For information about aFleX events, see aFleX Events.

For information about RAM caching commands, see RAM Caching Commands.

CACHE_REQUEST

Description Execute specific aFleX commands when a virtual server receives a request for a cached object.

Syntax

```
when CACHE_REQUEST { <aFleX commands> }
```

Usage Valid with the following RAM caching commands:

- CACHE::age
- CACHE::disable
- CACHE::enable
- CACHE::expire
- CACHE::headers
- CACHE::hits

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following global commands:

- active_members
- b64decode
- b64encode

- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to revalidate a cached object from the server if the age of the cache is greater than 60 seconds. A log message is also created:

```
when CACHE_REQUEST {
    if { [CACHE::age] > 60 } {
        CACHE::expire
        log "Expired Content: Age is greater than 60 seconds"
    }
}
```

CACHE_RESPONSE

Description Execute specific aFleX commands immediately before sending a cache response.

Syntax

```
when CACHE_RESPONSE { <aFleX commands> }
```

Usage Valid with the following RAM caching commands:

- CACHE::age
- CACHE::disable
- CACHE::enable
- CACHE::expire
- CACHE::headers
- CACHE::hits

Valid with the following class list commands:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit

- LID::type

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch

- use
- virtual

Example Use the following example to revalidate a cached object from the server if the `::expired` variable is set to 1. An expiration message is logged, and then the ::expired variable is set to 0.

```
when CACHE_RESPONSE {
        if { $::expired == 1 } {
                CACHE::expire
                log "cache expire"
                table set expired 0 0
        }
}
```

SIP Events

Session Initiation Protocol (SIP) events are supported for the following:

- SIP – Session Initiation Protocol over UDP
- SIP-TCP – SIP over TCP
- SIPS – Secure SIP over TLS

NOTE: For previous releases, only SIP over UDP is supported.

The following SIP events are available:

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

For information about aFleX events, see aFleX Events.

For information about SIP commands, see SIP Commands.

SIP_REQUEST

Description Execute specific aFleX commands when a full SIP request header is received from the client.

Syntax

```
when SIP_REQUEST { <aFleX commands> }
```

Usage Valid with the following SIP commands:

- SIP::call_id
- SIP::from
- SIP::header
- SIP::method
- SIP::respond
- SIP::response
- SIP::to
- SIP::uri
- SIP::via

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons

- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- snat
- snatpool
- string map
- substr
- switch
- use
- virtual

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Example Use the following example to log the value of the Call-ID whenever an SIP request header is received.

```
when SIP_REQUEST {
    log "SIP Call_ID: [SIP::call_id]"
}
```

SIP_REQUEST_SEND

Description Execute specific aFleX commands when a SIP request is sent to the server.

Syntax

```
when SIP_REQUEST_SEND { <aFleX command> }
```

Usage Valid with the following SIP commands:

- SIP::call_id
- SIP::from

- SIP::header
- SIP::method
- SIP::respond
- SIP::response
- SIP::to
- SIP::uri
- SIP::via

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return

- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Valid with the following IP commands:

- IP::addr
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port

- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Example Use the following example to log the SIP method type whenever the ACOS device sends a SIP request to the server.

```
when SIP_REQUEST_SEND {
    log "SIP Method: [SIP::method]"
}
```

SIP_RESPONSE

Description Execute specific aFleX commands when a full SIP response is received from the server.

Syntax

```
when SIP_RESPONSE { <aFleX commands> }
```

Usage Valid with the following SIP commands:

- SIP::call_id
- SIP::from
- SIP::header
- SIP::method
- SIP::respond
- SIP::response
- SIP::to
- SIP::uri
- SIP::via

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage

- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- snat
- snatpool
- string map
- substr
- switch
- use
- virtual

Valid with the following IP commands:

- IP::addr
- IP::client_addr

- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::server_addr
- IP::tos
- IP::ttl
- IP::version

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::option
- TCP::rtt
- TCP::server_port

Related Information

- TCP::remote_port
- TCP::respond

Valid with the following UDP commands:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::server_port

Example Use the following example to log the SIP response code whenever a full SIP response from the server is received.

```
when SIP_RESPONSE {
    log "SIP Response Code: [SIP::response code]"
}
```

SMTP Events

The following Financial Information eXchange (FIX) events are available:

- SMTP_MAIL
- SMTP_EHLO

For information about aFleX events, see aFleX Events.

For information about FIX commands, see Financial Information eXchange Commands.

SMTP_MAIL

Description Triggers upon receiving MAIL FROM command from the client.

Syntax

```
when SMTP_MAIL { <aFleX commands> }
```

Usage Valid with the following SMTP commands:

- SMTP::mail

Example Use this example for routing SMTP traffic based on the sender's email domain.

```
when SMTP_MAIL {
        If {[SMTP::mail] equals abc.com} {
                node 1.1.1.1 25
                        } else {
                                Node 2.2.2.2 25
                                }
                }
```

SMTP_EHLO

Description Triggered when EHLO command arrives

Syntax

```
when SMTP_EHLO { <aFleX commands> }
```

Usage Valid with the following SMTP commands:

- SMTP::greet
- SMTP::ehlo

Example Use this example for routing SMTP traffic based on the sender's email domain.

```
when SMTP_EHLO {
                SMTP::greet "VRFY"
        }
```

SSL Events

The following SSL events are available:

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERCERT
- SERVERSSL_SERVERHELLO

For information about aFleX events, see aFleX Events.

For information about SSL commands, see SSL Commands.

CLIENTSSL_CLIENTCERT

Description Execute specific aFleX commands when an SSL client certificate is received.

Syntax

```
when CLIENTSSL_CLIENTCERT { <aFleX commands> }
```

Usage Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::exists
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following SSL commands:

- SSL::authenticate
- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::template

- SSL::verify_result

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop

- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- session
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to create a log and set the subject of the log entry when an SSL client certificate is received.

```
when CLIENTSSL_CLIENTCERT {
    log "X509 Subject: [X509::subject [SSL::cert 0]]"
}
```

CLIENTSSL_CLIENTHELLO

Description Execute specific aFleX command when an SSL Client Hello message is received.

Syntax

```
when CLIENTSSL_CLIENTHELLO { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::authenticate
- SSL::cert
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::template

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage

- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to begin collecting SSL application data when an SSL Client Hello message is received:

```
when CLIENTSSL_CLIENTHELLO {
    SSL::collect 100
}
```

```
when CLIENTSSL_DATA {
    log "SSL Payload Length: [SSL::payload length]"
    log "SSL Payload: [SSL::payload]"
    SSL::release
}
```

CLIENTSSL_DATA

Description Execute specific aFleX commands when the ACOS device is in SSL collect mode and receives an SSL application data message from a client.

Syntax

```
when CLIENTSSL_DATA { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::authenticate
- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::renegotiate
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port

- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield

- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use this example to trigger SSL authentication and renegotiation when the device enters SSL collect mode and receives SSL application data.

```
when CLIENT_ACCEPTED {
    set renegotiate 1
    set index 1
}
when CLIENTSSL_HANDSHAKE {
    if { $renegotiate == 1 } {
        log "SSL Handshake done - Index: $index"
        incr index
        set renegotiate 0
        SSL::collect
    } else {
```

```
        log "SSL Renegotiate Handshake done - Index: $index"
        incr index
        SSL::release
    }
}
when CLIENTSSL_DATA {
    log "Start SSL Renegotiate - Index: $index"
    SSL::authenticate depth 2
    SSL::authenticate once
    SSL::cert mode request
    SSL::renegotiate
}
```

CLIENTSSL_HANDSHAKE

Description Execute specific aFleX commands when an SSL handshake on the client side is completed.

Syntax

```
when CLIENTSSL_HANDSHAKE { <aFleX commands> }
```

Usage Valid with the following limit ID commands:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::exists
- LID::request_limit
- LID::request_rate_limit
- LID::type

Valid with the following SSL commands:

- SSL::authenticate
- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable

- SSL::mode
- SSL::payload
- SSL::release
- SSL::renegotiate
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::sessionsecret
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text

- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map

- substr
- switch
- use
- virtual

Example Use the following example to create a log and set the subject of the log entry whenever an SSL handshake is completed on the client side.

```
when CLIENTSSL_HANDSHAKE {
    log "X509 Subject: [X509::subject [SSL::cert 0]]"
}
```

SERVERSSL_CLIENTHELLO_SEND

Description Execute specific aFleX commands when the ACOS device sends a SSL Client Hello message to the back-end server.

Syntax

```
when SERVERSSL_CLIENTHELLO_SEND { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::authenticate
- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::renegotiate
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain

- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to begin collecting SSL application data whenever an SSL Client Hello message is sent to the server.

```
when SERVERSSL_CLIENTHELLO_SEND {
    SSL::collect
}
when SERVERSSL_DATA {
    log "SSL Payload Length: [SSL::payload length]"
```

```
    log "SSL Payload: [SSL::payload]"
    SSL::release
}
```

SERVERSSL_DATA

Description Execute specific aFleX commands when ACOS device is in SSL collect mode and receives an SSL application data message from a back-end server.

Syntax

```
when SERVERSSL_DATA { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::respond
- SSL::sessionid
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log

- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to log SSL data information and release the collected data whenever the ACOS Device enters SSL collect mode and receives data from the back-end server.

```
when SERVERSSL_HANDSHAKE {
    SSL::collect 400
}
when SERVERSSL_DATA {
    log "SSL Payload Length: [SSL::payload length]"
    log "SSL Payload: [SSL::payload]"
    SSL::payload replace 0 [SSL::payload length] "HTTP/1.1 200
OK\r\nContent-Length: 37\r\nContent-Type:
text/html;\r\n\r\n<html><head>Hello World!</head></html>"
    SSL::release
}
```

SERVERSSL_HANDSHAKE

Description Execute specific aFleX commands when an SSL handshake on the server side is completed.

Syntax

```
when SERVERSSL_HANDSHAKE { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::respond
- SSL::sessionid
- SSL::sessionsecret
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer

- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop

- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to create a log and set the subject of the log entry whenever an SSL handshake is completed on the server side.

```
when SERVERSSL_HANDSHAKE {
    log "X509 Subject: [X509::subject [SSL::cert 0]]"
}
```

SERVERSSL_SERVERCERT

Description Triggered when the device receives an SSL certificate from the server (after verification) .

Syntax

```
when SERVERSSL_SERVERCERT { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSLI::cache_cert
- SSL::cert
- SSLI::bypass
- SSLI::drop
- SSL::hostname
- SSLI::inspect

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Example Use the following example:

```
when SERVERSSL_SERVERCERT {
  set cert [SSL::cert 0]
  set text [X509::text $cert]
  log "SERVERSSL_SERVERCERT: text=$text"
}
```

SERVERSSL_SERVERHELLO

Description Execute specific aFleX command when an SSL Server Hello is received from a back-end server.

Syntax

```
when SERVERSSL_SERVERHELLO { <aFleX commands> }
```

Usage Valid with the following SSL commands:

- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSL::enable
- SSL::mode
- SSL::payload
- SSL::release
- SSL::respond
- SSL::sessionid
- SSL::template
- SSL::verify_result

Valid with the following application firewall commands:

- APPCLS::application

Valid with the following TCP commands:

- TCP::client_port
- TCP::local_port
- TCP::mss
- TCP::rtt

Valid with the following X509 commands:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

Valid with the following global commands:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop

- encoding
- event
- findstr
- getfield
- htonl
- htons
- if
- log
- md5
- members
- nexthop
- ntohl
- ntohs
- persist
- reject
- return
- serverside
- set encode
- sha1
- string map
- substr
- switch
- use
- virtual

Example Use the following example to enable SSL collect mode whenever an SSL Server Hello message is received:

```
when SERVERSSL_SERVERHELLO {
    SSL::collect
}
when SERVERSSL_DATA {
    log "SSL Payload Length: [SSL::payload length]"
    log "SSL Payload: [SSL::payload]"
```

```
    SSL::release
}
```

aFleX Commands

- RAM Caching Commands
- Resolve Commands
- SIP Commands
- SMTP Commands
- SSL Commands
- Statistics Commands
- Table Commands
- TCP Commands
- Template Commands
- Time Commands
- UDP Commands
- URI Commands
- URL Commands
- X509 Commands
- Deprecated and Disabled Commands

Overview

aFleX commands can perform the following types of operations:

- Global – Performs actions such as selecting a pool (SLB service group) or node (server).

Query commands:

- IP packet header query – Returns information from the IP header.
- IP, TCP, or UDP packet data query – Returns information from the payload.
- HTTP packet header or content query – Returns information from the HTTP header or payload.
- Header and content manipulation:
- HTTP cookie manipulation – Changes cookies.
- TCP header and content manipulation – Changes TCP headers or content.
- HTTP header and content manipulation – Changes HTTP headers or content.
- SSL and X.509 query – Returns information from or about certificates.
- Deep packet inspection – Returns strings from packets.

For information about other script components, see aFleX Script Components.

Global Commands

The following global aFleX commands are available:

- active_members
- b64decode
- b64encode
- clientside
- cpu usage
- domain
- drop
- encoding
- event
- findstr
- getfield
- htonl
- htons
- log
- lwnode
- md5
- members
- nexthop
- lwnode
- ntohl
- ntohs
- persist
- pool

- reject
- return
- serverside
- session
- set encode
- sha1
- snat
- snatpool
- string map
- substr
- switch
- use
- virtual
- when
- whereis

For information about aFleX commands, see aFleX Commands.

For information about global variable commands, see Global Variable Commands.

active_members

Description This command returns either the number of active members in a service group or pool or a listing. When the optional list parameter is not used, then the default behavior outputs the number of active members.

Syntax

```
active_members [list] {<pool_name> | <service-group>}
```

Example The following configuration checks the number of active members in the service group example_service_group. If the number of active members is 3 or fewer, traffic is directed to the backup pool service_group_backup :

```
when HTTP_REQUEST {
        if { [active_members example_service_group] <= 3 } {
        pool service_group_backup
}
}
```

Example The following configuration logs the list of currently active members in the service_group_http service group when an HTTP request is received:

```
when HTTP_REQUEST {
log "The service group http active member list is [active_
members list service_group_http]"
}
Output:
[AFLEX]:af: The service group http active member list is
{192.168.0.0 80}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

b64decode

Description This command returns a specified string that was decoded from base-64. If there is an error, it will return NULL.

Syntax

```
b64decode <string>
```

Example Use this example to decode a base-64 encoded cookie from the HTTP request:

```
when HTTP_REQUEST {
    set encoded_cookie [HTTP::cookie "EncodedCookie"]
    set decoded_cookie [b64decode $encoded_cookie]
    HTTP::cookie insert name "ClearCookie" value $decoded_
cookie
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

b64encode

Description This command returns a specified encoded base-64 string when used. If there is an error, it will return NULL.

Syntax

```
b64encode <string>
```

Example Use this example to Base64-encode a cookie from the HTTP request:

```
when HTTP_REQUEST {
    set decoded_cookie [HTTP::cookie "ClearCookie"]
    set encoded_cookie [b64encode $decoded_cookie]
    HTTP::cookie insert name "EncodedCookie" value $encoded_
cookie
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

b64urldecode

Description This command returns a specified string that was decoded from base-64URL. It handles URL-safe characters and optional padding. If there is an error, it will return NULL.

Syntax

```
b64urldecode <string>
```

Example Use this example to decode a Base64 URL-encoded string inside an HTTP request:

```
when HTTP_REQUEST {
    set encoded_data "SGVsbG8tV29ybGRf"  ;# URL-safe Base64
for "Hello-World_"
    set decoded_data [b64urldecode $encoded_data]
    log "Decoded result: $decoded_data"
}
```

For information about aFleX events, see aFleX Events.

b64urlencode

Description This command returns a specified base-64URL-encoded string. If there is an error, it will return NULL.

Syntax

```
b64urlencode <string>
```

Example Use this example to encode a string for safe transmission in URLs or tokens:

```
when HTTP_REQUEST {
    set plain_text "Hello-World_"
    set encoded_text [b64urlencode $plain_text]
    log "Encoded result: $encoded_text"
}
```

For information about aFleX events, see aFleX Events.

client_addr

Description This command retrieves the IP address of the client that initiated the connection. It is used to perform operations based on the client's address, such as filtering or redirecting traffic.

Syntax

```
client_addr { <aFleX commands> }
```

Example Use this example to log or redirect traffic based on client IP address:

```
when HTTP_REQUEST {
    if {[client_addr] eq "10.10.10.10"} {
        HTTP::redirect "http://restricted.example.com"
    } else {
        log "Client IP: [client_addr]"
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

client_port

Description This command retrieves the port number of the client making the connection. It is used to inspect or filter traffic based on the client's source port.

Syntax

```
client_port { <aFleX commands> }
```

Example Use this example to log the source port of a client making an HTTP request:

```
when HTTP_REQUEST {
    set src_port [client_port]
    log "Client is connecting from port: $src_port"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

clientside

Description Using this command will take specified aFleX commands to be put in assessment under the client-side context. It will not affect aFleX

commands that are already under the client-side context being assessed.

Syntax

```
clientside { <aFleX commands> }
```

Example

The following example uses the clientside command to retrieve the client-side IP address after the server connection is established. If the client's IP address matches 192.168.0.0, the connection is discarded:

```
when SERVER_CONNECTED {
    if { [IP::addr [clientside {IP::remote_addr}] equals
192.168.0.0.0] } {
        discard
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

cpu usage

Description

This command will return the average CPU load for an interval based on the defined time. The average is a moving average that is exponentially weighted over an interval.

Syntax

```
cpu usage [1sec | 5secs | 15secs | 1min | 5mins | 15mins |
all_seconds | all_minutes]
```

Example

The following example uses the `cpu` command to check the average CPU load over the last 15 seconds:

```
when HTTP_REQUEST {
    if { [cpu usage 15secs] <= 60} {
example_service_group
    } else {
        HTTP::redirect "http://backup.example.com"
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

discard

Description Depending on the event, this command will discard the connection or current packet. This must be conditionally associated with an if statement and essentially functions the same as the drop command.

Syntax

```
discard
```

Example Use the following example to discard the connection if the client's IP address matches 192.168.0.0:

```
when SERVER_CONNECTED {
    if { [IP::addr [IP::remote_addr] equals 192.168.0.0] } {
        discard
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

dnat

Description Enables destination NAT on the current connection by modifying the destination IP address. Commonly used to redirect traffic to internal servers.

Syntax

```
dnat { <aFleX commands> }
```

Example Use this example to change the destination IP address for an incoming HTTP request to 192.168.1.100:

```
when HTTP_REQUEST {
    dnat 192.168.1.100
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

domain

Description This command returns a specified string as a dotted domain name. In addition, the last <count> portions of a domain name will be returned.

Syntax

```
domain <string> <count>
```

Example Use this example to route traffic to `example_service_group` if the top-level domain in the HTTP host is `com`:

```
when HTTP_REQUEST {
    if { [domain [HTTP::host] 1] equals "com" } {
        pool example_service_group
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

drop

Description Depending on the event, this command will drop the connection or current packet. This must be conditionally associated with an if statement and essentially functions the same as the discard command.

Syntax

```
drop
```

Example Use the following example to drop the connection if the client IP address is 192.168.1.10:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10] } {
        drop
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

encoding

Description This command takes a character encoded payload and converts it to the specified encoding format.

Syntax

```
encoding {convertfrom | convertto} <encoding>
```

Example See set encode.

Valid Events

All

For information about aFleX events, see aFleX Events.

esha256

Description Generates a digital signature using ECDSA (Elliptic Curve Digital Signature Algorithm) combined with the SHA-256 hashing algorithm.

Syntax

```
esha256 { <aFleX commands> }
```

Example Use the following example to generate and log an ECDSA SHA-256 signature for a string when a client connects:

```
when CLIENT_ACCEPTED {
    log "[esha256 "123456789"]"
```

```
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

event

Description This command will abandon the review of specified aFleX events, or everything on the connection depending on the parameters chosen, while the aFlex script continues to run.

Syntax

```
event [<name>] [enable | disable] | [enable all | disable all]
```

Example Use the following example to disable the HTTP_REQUEST event when the client IP address is 192.168.0.0:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.0.0] } {
        event HTTP_REQUEST disable
    }
}
when HTTP_REQUEST {
    log "There is a HTTP Request from: [IP::client_addr]"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

findstr

Description This command is used to locate a string <search_string> within another string <string>, where the result is the offset string specified from the match. The value for <terminator> is a character or number (length). If not specified, it will default to the end of the string. <Skip_count> will default to zero if not specified. If neither parameters <terminator> and

<skip_count> are defined, it functions as the command `string range <string> [string first <string> <search_string>] end`.

Syntax

```
findstr <string> <search_string> [<skip_count> [<terminator>]]
```

Example

The following example checks if the URI contains "type=" and returns "cgi" after skipping 5 characters, then directs traffic to different pools based on the result:

```
when HTTP_REQUEST {
    if { [findstr [HTTP::uri] "type=" 5 "&"] eq "cgi" } {
        pool service_group_dynamic
    } else {
        pool example_service_group
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

forward

Description

Sends the current packet or connection to its next destination based on current routing or SLB decisions. This command is used to manually resume or continue forwarding after a conditional check or custom logic in the script.

Syntax

```
forward { <aFleX commands> }
```

Example

Use this example to forward the request to node 1.1.1.1 if certain conditions are met:

```
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/api" } {
        forward node 1.1.1.1
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

getfield

Description This command provides the corresponding string from a specified field through the <string> or <split> attributes.

Syntax

```
getfield <string> <split> <field_number>
```

Example Use the example to show the extraction of the hostname from the host header.

```
when HTTP_REQUEST {
        [getfield [HTTP::host] ":" 1]
}
```

Example Use the second example to show how to redirect request for `example.net` to `example.edu`

```
when HTTP_REQUEST {
    if { [HTTP::host] contains "example.net" } {
        HTTP::redirect http://[getfield [HTTP::host]
".example.net" 1].example.edu[HTTP::uri]
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

hsha256

Description Generates a digital signature using HMAC (Hash-based Message Authentication Code) combined with the SHA-256 hashing algorithm.

Syntax

```
hsha256 { <aFleX commands> }
```

Example Use the following example to generate and log the HMAC-SHA256 hash of the data using a secret key when a client connects:

```
when CLIENT_ACCEPTED {
    set key "mysecret"
    set message "hello-world"
    log "HMAC-SHA256: [hsha256 $key $message]"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

htonl

Description This command converts a hosts' byte order of an unsigned integer to network byte order.

Syntax

```
htonl <hostlong>
```

Example The following example converts the integer 12348765 from host byte order to network byte order:

```
when HTTP_REQUEST {
        set hostlong 12348765
        set netlong [htonl $hostlong]
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

htons

Description This command converts a hosts' byte order of an unsigned short integer to network byte order.

Syntax

```
htons <hostshort>
```

Example

The following example converts the integer 1423 from host byte order to network byte order (16-bit) using the htons command:

```
when HTTP_REQUEST {
        set hostshort 1423
        set netshort [htons $hostshort]
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

if

Description

Use this command to query for a true or false answer, and take action based upon that answer. The *elseif* and else commands can be added after an *if* command.

A recognition of the initial *if* statement as false will mean the evaluation of *elseif*. If *elseif* is assessed as true, its command is executed, but if it is assessed as false, the command associated with else will run.

Use of multiple *elseif* statements is allowable from a single *if* statement.

Syntax

```
if { <expression> } { <statement_command> }
elseif { <expression> } { <statement_command> }
else { <expression> } { <statement_command> }
```

NOTE: The allowable maximum number of *if* statements that can be nested in an aFleX policy is 100.

Example

Use this example for routing HTTP requests to different service groups based on the file extension in the URI:

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
```

```
        pool service_group_static
    } elseif { [HTTP::uri] ends_with ".asp" } {
        pool service_group_dynamic
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

ip_protocol

Description Retrieves the IP protocol value from the current packet. Can be used for conditional logic within aFleX scripts to match specific protocols (e.g., TCP, UDP, ICMP). Must be conditionally associated with an if statement.

Syntax

```
ip_protocol { <aFleX commands> }
```

Example Use this example to check if the IP protocol is UDP (protocol number 17), and take an action:

```
when CLIENT_DATA {
    if { [ip_protocol] == 17 } {
        log "UDP traffic detected"
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

ip_tos

Description Retrieves the Type of Service (ToS) value from the IP header of the packet. Often used for traffic classification or prioritization.

Syntax

```
ip_tos { <aFleX commands> }
```

Example Use this example to log traffic with a specific ToS value:

```
when CLIENT_DATA {
    if { [ip_tos] == 184 } {
        log "High-priority traffic detected"
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

ip_ttl

NOTE: The `ip_ttl` command is deprecated. Instead, use the recommended equivalent `IP::ttl`. While `ip_ttl` remains functional, switching to `IP::ttl` is recommended.

Description Retrieves the Time to Live (TTL) value from the IP header of the packet. TTL helps determine how many hops a packet can traverse before being discarded.

Syntax

```
ip_ttl { <aFleX commands> }
```

Example Use this example to drop packets with TTL less than or equal to 1:

```
when CLIENT_DATA {
    if { [ip_ttl] <= 1 } {
        drop
        log "Dropped packet with TTL <= 1"
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

log

Description This command creates and logs a specified message to the Syslog utility through a variable expansion on messages as prescribed for the HTTP profile Header Insert setting. Use "local0" to "local7" as the value for facility (Note: only "local0" is supported). For <level>, the number value from 0 to 7 can be used, or its corresponding level string, "EMERG", "ALERT", "CRIT", "ERR", "WARNING", "NOTICE", "INFO", and "DEBUG".

Syntax

```
log [<facility>.<level>] <message>
```

NOTE:

- Use the log command carefully, as it can produce an enormous amount of input.
- When using the Syslog facility, the log is limited to 1024 bytes per request. Longer strings will be truncated.
- Regardless of level, the aFlex log command messages are rate limited as a class. Thus, subsequent messages within the rate-limit period may be curbed despite textual differences. Duplicate messages in Syslog are suppressed.

Example The following example logs a message using the default facility local0 and default level INFO (6):

```
log "The log message is from facility local0 by default and
level INFO (6) by default"
```

Example The following example logs a message using facility local2 and default level INFO (6):

```
log local2."The log message is from facility local2 and level
INFO (6) by default"
```

Example The following example logs a message using facility local2 and severity level 0 (EMERG):

```
log local2.0 "This log massage is from facility local2 and
level 0 (EMERG)"
```

Example The following example logs a message using facility local2 and severity level DEBUG (7):

```
log local2.DEBUG "This log massage is from facility local2 and
level DEBUG (7)"
```

NOTE: In ACOS 4.0.1, aFlex log entries are recognized so log messages for aFlex events will be linked with the aFlex script where they occurred.

Example In ACOS 4.0.1 and higher, with the application of three aFleX scripts (af1, af2, and af3) to its virtual port using the `show log` output for an SLB, virtual server will display the following:

```
ACOS(config)#show log
Aug 05 2014 11:58:14 Info    [AFLEX]:af3:HTTP status : 200
Aug 05 2014 11:58:14 Info    [AFLEX]:af3:HTTP_RESPONSE event
Aug 05 2014 11:58:14 Info    [AFLEX]:af2:Another http request
cmd!
Aug 05 2014 11:58:14 Info    [AFLEX]:af1:This is http_request_
1
```

Prior to ACOS 4.0.1, the application of the three aFlex scripts would have produced the following `show log` output:

```
ACOS(config)#show log
Aug 14 2014 10:11:07 Info    [AFLEX]:af1+af2+af3:HTTP status :
200
Aug 14 2014 10:11:07 Info    [AFLEX]:af1+af2+af3:HTTP_RESPONSE
event
Aug 14 2014 10:11:07 Info    [AFLEX]:af1+af2+af3:Another http
request cmd!
Aug 14 2014 10:11:07 Info    [AFLEX]:af1+af2+af3:This is http_
request_1
```

Valid Events

All

For information about aFleX events, see aFleX Events.

lwnode

Description This command forces the specified server node to be used directly, bypassing any load-balancing. The difference between this command and the `node` command is that `lwnode` can be referred to an entity that is not a service-group member or a defined part of the real server configuration.

Syntax

```
lwnode <addr> [<port>]
```

NOTE: When using the `lwnode` command, a source NAT pool must be applied.

NOTE: Nodes selected by using this command do not have connection limiting and connection rate limiting applied.

Example Use this example to route incoming HTTP requests to different origin servers based on the URI path:

```
when HTTP_REQUEST {
    switch -glob [string tolower [HTTP::uri]] {
"/static1/*" { lwnode 192.168.0.1 }
        "/static2/*" { lwnode 192.168.0.2 }
        "/static3/*" { lwnode 192.168.0.3 }
        default { lwnode 192.168.0.4 8080 }
}
}
```

Valid Events c

md5

Description This command provides the RSA MD5 Message Digest Algorithm message digest of the specified string.

Syntax

```
md5 <string>
```

Example The following example generates the MD5 hash of a string and converts it to a readable ASCII format:

```
when CLIENT_ACCEPTED {
```

```
    set md5_binary [md5 "1234509876"]
    binary scan $md5_binary H* md5_ascii
    log "MD5: $md5_ascii"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

members

Description This command counts or lists all the service group members.

Syntax

```
members [list] <pool>
```

Example Use the following example to list members. If this option is removed, the output is the member count.

```
when CLIENT_ACCEPTED {
    log "Here are the Total Member(s): [members list example_
service_group]"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

nexthop

Description This command will set the next hop for a connection.

For the events listed, using this command overwrite the default reverse next-hop IP address:

- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

- SERVER_CONNECTED
- SERVER_DATA

For other cases, use of this command will overwrite the forward next-hop IP address.

Syntax

```
nexthop <ipaddr>
```

Example

Use the following example to configure conditional next-hop routing based on the client's IP address:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.0/24] }
{
        nexthop 192.168.1.254
        log "Nexthop: 192.168.1.254"
    } else {
        log “Nexthop: default (192.168.1.10)”
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

node

Description

This command forces the specified server node that is comprised of an IP address and a port number to be used directly, and bypass any load-balancing.

NOTE:

This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
node <addr> [<port>]
```

NOTE:

- Use of the `node` command requires the configuration of a real server (node) and service port as a member of a service group.
- Nodes selected through this command do not have connection limiting and connection rate limiting applied to them.

Example Use the following example to send an HTTP request to a specific node with the IP address 192.168.1.200 on port 80 for a URI ending with '.png'.

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".png" } {
        node 192.168.1.200 80
    }
}
```

ntohl

Description This command converts a network byte order's unsigned integer to a host byte order.

Syntax

```
ntohl <netlong>
```

Example The following example converts the network-ordered integer 12348765 to host byte order:

```
when HTTP_REQUEST{
        set netlong 12348765
        set hostlong [ntohl $netlong]
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

ntohs

Description This command converts a network byte order's unsigned short integer to a host byte order.

Syntax

```
ntohs <netshort>
```

Example The following example converts the network-ordered short integer 1243 to host byte order:

```
when HTTP_REQUEST {
        set netshort 1243
        set hostshort [ntohs $netshort]
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

persist

Description This command sets client persistence based on the value chosen.

Syntax

```
persist uie <string> [<timeout>]
```

Use of this command sets the key for an entry on the persistence table. This maps the client to an SLB resource (real server, real server port, or service group). If the persistence table contains the specified key, the ACOS device uses the SLB resource that key is mapped to in the table. Otherwise, the ACOS device will use SLB to select a resource and create a corresponding persistence table entry. The `uie` option, "Universal Inspection Engine", indicates that persistence can be set based on any key.

The `<timeout>` parameter specifies how many seconds the persistence entry can remain in the table after last time key from the client is sent to the server. The default is 1800 seconds and the max timeout is 15240 seconds. Internally, the timeout is converted to minutes and is decremented one minute at a time.

Use the following syntax to ignore server template limits for persistence server selection.

```
persist uie <string> [<timeout>] [dont_honor_conn_rules]
```

Use the following syntax to add an entry to the persistence table. This command differs from the command above because it does not first check the persistence table for an existing entry for the key. The `persist add` form of the command is useful for setting persistence based on data that is set on the server and is therefore first observed by the ACOS device in the server response, rather than in the client request.

```
persist add uie <key> [<timeout>]
```

Use the following syntax to perform a lookup in the persistence table for an entry with the specified key:

```
persist lookup uie <key> [all | node | port | pool]
```

- `all` – This parameter returns all the values listed below. (If not specified, and none of the other options are specified, the command interpretation is equivalent to specifying `all`.
- `node` – This parameter will return the real server IP address.
- `port` – This parameter will return the real service port number.
- `pool` – This parameter will return the pool (service group) name.

Use the following syntax to delete the persistence table entry for the specified key.

```
persist delete uie <key>
```

Use the following syntax to return the number of persistent entries corresponding to a virtual port. If global is specified, the command will provide the number of persistent entries in the entire partition.

```
persist size uie [global]
```

Syntax

```
<key>
```

The use of the `<key>` specifies the data upon which the persistence is based. It can be specified with one of the following options:

```
<specified-value>
```

Use persist to the same real server and port if traffic contains the specified key value and is sent to the same virtual port.

```
{<specified-value> [any virtual | any service | any pool]
[pool <pool-name>]}
```

These options provide the following behaviors:

`<specified-value>`	Key value
`any virtual`	Use for persistence to the same real server and port if traffic contains the specified key value and is sent to the same virtual port and service group (pool).
`any service`	Use for persistence to the same real server if traffic contains the specified key value and is sent to the same virtual server, to any virtual port.
`any pool`	Use for persistence to the same real server if traffic contains the specified key value.
`pool <pool-name>`	Use for persistence to the same real server and port if traffic contains the specified key value and is sent to the same virtual port and to the specified service group.

NOTE:

- Server template limits are applied for both service-group and server selection. Commands that call for server selection such as “node”, “pool”, and “persist” will enforce server template limits on the selected server. As a result, new connections that match a persist uie entry may find themselves unable to use the rport and a default server selection will occur instead. To prevent default server selection, use the no def-selection-if-pref-failed command for the vport.
- If the length of the persist UIE key in aFlex exceeds the internal limit of 63 characters, aFlex truncates the key to an appropriate length for use.
- To show the persistent sessions managed by this aFleX command, use the following command in the CLI: *show session persist uie.*

Example Use the following example script to provide persistence on a VIP on any port.

```
when HTTP_REQUEST {
    set IP [IP::client_addr]
    set p [persist lookup uie { $IP any virtual } all]
    if { $p ne "" } {
        log " UIE located ([lindex $p 0] [lindex $p 1] [lindex
$p 2])"
        node [lindex $p 1] [lindex $p 2]
    }
}
when HTTP_RESPONSE {
    set IP [IP::client_addr]
    persist add uie { $IP any virtual } 1800
}
```

Example Use the following example script to provide the same persistence for a client IP address accessing one VIP and port:

```
when HTTP_REQUEST {
    set IP [IP::client_addr]
    persist uie $IP
}
```

```
when HTTP_RESPONSE {
    set IP [IP::client_addr]
    persist add uie $IP 1800
}
```

Example Use the following example script to provide the same persistence for a client IP address accessing any VIP and any port:

```
when HTTP_REQUEST {
    set IP [IP::client_addr]
    set p [persist lookup uie { $IP any service } all]
    if { $p ne "" } {
        log " UIE located([lindex $p 0] [lindex $p 1] [lindex
$p 2])"
        node [lindex $p 1] [lindex $p 2]
    }
}
when HTTP_RESPONSE {
    set IP [IP::client_addr]
    persist add uie { $IP any service } 1800
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

pool

Description This command will cause the system to load balance traffic to the specified pool or pool member. This statement must have an if statement conditionally associated with it.

This command acts upon the service groups (pools) located in the partition that contains the aFleX policy.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
pool <pool_name>
```

```
pool <pool_name> [member <addr> [<port>] ]
```

NOTE:

- Pool/member may be selected conditionally. If multiple conditions match, the last match determines the pool/member to which this traffic is load balanced.
- Server template limits are applied for both service-group and server selection. Commands that call for server selection such as `node`, `pool`, and `persist` will enforce server template limits on the selected server. As a result, new connections that match a persist uie entry may be unable to use the rport and a default server selection will occur instead. To prevent a default server selection, use the no def-selection-if-pref-failed command for the vport.

Example

The following example routes traffic from a specific client IP to a designated pool:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10] } {
        pool example_server_group
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- LB_FAILED

Events that do not generate an error, but are likely not valid for this command:

- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_SELECTED

- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

reject

Description This command will cause the connection to be rejected, and return a reset as appropriate for the protocol.

Syntax

```
reject
```

NOTE: When the command `reject` is used during the `AAM_AUTHORIZATION_CHECK` event, the authentication session is not established.

Example The following example rejects the connection if the client IP address matches 192.168.1.10:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10] } {
        reject
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

rsha256

Description Generates a digital signature using RSA with PKCS#1 v1.5 padding combined with the SHA-256 hashing algorithm.

Syntax

```
rsha256 { <aFleX commands> }
```

Example Use the following example to generate and log an RSA SHA-256 signature for a string when a client connects:

```
when CLIENT_ACCEPTED {
    log "[rsha256 "123456789"]"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

return

Description This command terminates execution of an aFleX event and would optionally return the result of the evaluating expression.

Syntax

```
return [<expression>]
```

Example Use of the following example shows that the foreach loop is broken by the `return` command if the string “X-ClientIP” is found in the HTTP header.

```
when HTTP_REQUEST {
    foreach header [HTTP::header names] {
        if { [HTTP::header exists "X-ClientIP"] } {
            return
        } else {
            HTTP::header insert X-Forwarded-For [IP::client_
addr]
        }
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

serverside

Description This command will cause the specified aFleX command or commands to be evaluated under the server-side context. This command has no effect if the aFleX policy is already being evaluated under the server-side context.

Syntax

```
serverside { <aFleX command> }
```

Example The following example drops the connection if the remote IP address of the server matches 192.168.80.81:

```
when CLIENT_ACCEPTED {
    if {[IP::addr [serverside {IP::remote_addr}] equals
192.168.80.81] } {
        drop
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

session

Description This command manages SSL sessions.

Syntax

```
session <add|lookup|delete> ssl <key>
```

Use the following syntax to create a table to store SSL information. If an SSL table already exists, the command will add an entry to the table. Generally, the `<key>` is the session ID and the data is the SSL verify_ result or the SSL certificate.

```
session add ssl <key> <data> [<timeout>]
```

Use the following syntax to search the SSL table for information about the specified key:

```
session lookup ssl <key>
```

Use the following syntax to delete a specified SSL entry:

```
session delete ssl <key>
```

Example Use the following example to store client certificate against the SSL session ID during the handshake phase and then retrieve it:

```
when CLIENTSSL_HANDSHAKE {
        set cert1 [SSL::cert 0]
        session add ssl [SSL::sessionid] $cert1 300
}
when HTTP_REQUEST {
        set cert2 [session lookup ssl [SSL::sessionid]]
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_RESPONSE

set encode

Description This command will set the character encoding for data payloads.

Syntax

```
set encode "<encoding>"
```

Example Use the following example of an aFleX policy to convert payload data into Japanese encoding Shift_JIS:

```
when HTTP_RESPONSE {
        if { [HTTP::header "Content-Type"] contains "Shift_JIS" } {
                set encode "shiftjis"
                HTTP::collect
        }
}
when HTTP_RESPONSE_DATA {
        set hoge [HTTP::payload length]
        set payload [encoding convertfrom $encode [HTTP::payload]]
```

```
        regsub -all "abc" $payload "xyz" newdata
        set newdata3 [encoding convertto $encode $newdata]
        HTTP::payload replace 0 $hoge $newdata3
        HTTP::release
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

sha1

Description This command will return the Secure Hash Algorithm version 1.0 (SHA1) message digest of the specified string.

Syntax

```
sha1 <string>
```

NOTE: In case of an error, an empty string is returned.

Example Use this example to hash a user-supplied value and log the result:

```
when HTTP_REQUEST {
    set input "example_input"
    set hash [sha1 $input]
    log "SHA-1 hash of input: $hash"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

sha256

Description Generates a digital signature using SHA-256 (Secure Hash Algorithm version 2.0) hashing algorithm.

Syntax

```
sha256 <string>
```

Example Use the following example to generate and log the SHA-256 hash of a given string when a client connects:

```
when CLIENT_ACCEPTED {
    log "[sha256 "123456789"]"
}
```

NOTE: In the case of an error, an empty string is returned.

Valid Events

All

For information about aFleX events, see aFleX Events.

snat

Description This command will assign, select or disable source NAT.

Syntax

```
snat <addr>
```

Usage Use of this command will assign the specified NAT address (<addr>) to the server-side connection. The command replaces the reverse destination address of the connection with the specified IP address.

Example The following example script will apply the specified source NAT address for clients in the 192.168.10.0/24 subnet:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.10.0/24] }
{
        snat 203.113.80.150
    } else {
        snat 203.113.80.250
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- LB_SELECTED
- SIP_REQUEST

- SIP_RESPONSE
- DB_COMMAND
- DB_QUERY
- HTTP_REQUEST

snatpool

Description This command will use the specified pool of IP addresses as translation addresses to create a SNAT. It uses the specified NAT pool instead of the NAT pool that is already bound to the virtual port in the ACOS configuration.

Syntax

```
snatpool <snatpool_name>
```

The `<snatpool_name>` option will specify the name of a configured IP address pool.

```
snatpool none
```

The none option disables the SNAT.

NOTE:

- A NAT pool must already be bound to a virtual port in the ACOS configuration. This is the virtual port's default NAT pool.
- The IP type, IPv4 or IPv6 of the pool must be the same as the IP type of the real servers.

Example The following example assigns the source NAT pool based on the client's IP address. If the client's IP is 192.168.1.10, it uses the snat-internal pool; otherwise, it uses the snat-external pool:

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10] } {
        snatpool snat-internal
    } else {
        snatpool snat-external
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- HTTP_REQUEST
- LB_SELECTED
- SIP_REQUEST
- SIP_RESPONSE

For Layer 4 virtual ports, the `snatpool` command must be triggered by a `CLIENT_ACCEPTED` or `LB_SELECTED` event. In the case of Layer 7 ports, the `snatpool` command must be triggered by a HTTP_REQUEST event.

string map

Description This command will map the value of the second string to the value of the first string. Each instance of the `<string1>` will be replaced with `<string2>`.

Syntax

```
string map <string1> <string2>
```

Example This example illustrates that when an HTTP request comes in and has "/abc" in its uri, it will be changed to "/def" when it is sent to the backend server.

```
when HTTP_REQUEST {
    if {[HTTP::uri] contains "static"} {
HTTP::uri  [string map {"/static" "/images"} [HTTP::uri]]
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

substr

Description This command returns a sub-string named `<string>`, based on the values of the `<skip_count>` and `<terminator>` arguments.

Syntax

```
substr <string> <skip_count> [<terminator>]
```

The <skip_count> and <terminator> arguments are used in exactly the same fashion as they are for the findstr command.

The <skip_count> argument is the index into <string> of the first character to be returned where the value 0 will indicate the first character of <string>.

The <terminator> argument can either be the substring length or the substring terminating string. When it is an integer, the returned string includes that many characters, or up to the end of the string, whichever is shorter. When it is a string, the returned string includes characters up to but not including the first occurrence of the string. When it is a string which does not occur in the search space, from <skip_count> to the end of <string> is returned.

This command is the same as the Tcl string range command except that the value of the <terminator> argument may either be a character or a count.

```
when HTTP_REQUEST {
        set uri [substr $uri 1 "?"]
        log local0. "Uri Part = $uri"
}
log "[substr "abcdefghijklm" 2 "x"]"
log "[substr "abcdefghijklm" 2 "gh"]"
log "[substr "abcdefghijklm" 2 4]"
log "[substr "abcdefghijklm" 2 20]"
log "[substr "abcdefghijklm" 2 0]"
```

The above example logs the following:

```
cdefghijklm
cdef
cdef
cdefghijklm
cdefghijklm
```

Valid Events

All

For information about aFleX events, see aFleX Events.

switch

Description This is a built-in Tcl command that evaluates one of several scripts, depending on a given value.

Syntax

```
switch ?options? string {pattern body ?pattern body ...?}
```

This command matches its string argument against each of the pattern arguments in order. As soon as the command finds a pattern that matches the string, it evaluates the following body argument by passing it recursively to the Tcl interpreter and returns the result of that evaluation. If the last pattern argument is "default", then it matches anything. When no pattern argument matches string and no default is given, the command returns an empty string.

If the initial arguments start with "-", then these arguments are treated as options. The proceeding options are supported:

Use this following default option for exact matching to compare string to a pattern.

```
-exact
```

If matching string to the patterns, use the following option for glob-style matching which is the same as implementation by the string match command.

```
-glob
```

If matching string to patterns, use regular expression matching which is the same as implementation by the regexp command.

```
-regexp
```

This will mark the end of options. The argument following this one will be treated as string even if it starts with a "-".

```
--
```

There are two syntaxes provided for the pattern and body arguments.

The first syntax uses a separate argument for each of the patterns and commands. It is normally easier to use if substitutions are desired on some of the patterns or commands.

The second form will place all of the patterns and commands together into a single argument. The argument must have a proper list structure with elements of the list being the patterns and commands. The second form facilitates construction of multi-line commands since the braces around the whole list make it unnecessary to include a backslash at the end of each line. Since the second form has its pattern arguments in braces, no command or variable substitutions are performed on them; this differentiates the second form from the first form in some situations.

When a body is specified as "-", it means the body's next pattern should be used as the body for this pattern. Although, note that when the next pattern also has a body of "-", then the body after that is used, and so forth. Doing this allows sharing of a single body among several patterns.

NOTE: If the result of the `switch` evaluation is invalid, the script stops but no compilation error will be displayed. Make sure that all possible outcomes are valid, or consider using the `if ... elseif` syntax instead of `switch`.

In this example, the following script gives a compilation error, as expected:

```
when CLIENT_ACCEPTED {
     pool $invalid_server_group
}
```

However, the following script will not give a compilation error:

```
when CLIENT_ACCEPTED {
     $value = $somevalue
     switch $value {
          0 {
             pool $invalid_server_group
            }
            default {
            }
      }
}
```

Example The use of this example produces 2:

```
switch abc a - b {format 1} abc {format 2} default {format 3}
```

Example The use of this example produces a 3:

```
switch xyz {
        a
                -
        b
                {format 1}
        a*
                {format 2}
        default
                {format 3}
}
```

Example The use of the following example sends traffic with host header "www.domain.com" to pool www, host header "www.domain2.com", which will cause header manipulation and URI rewriting to take place first, and requests with any other host header will be discarded:

```
when HTTP_REQUEST {
    switch -glob [string tolower [HTTP::uri]] {
        "/images*" -
        "/static*" { pool service_group_static }
        "/blog*" { pool service_group_example }
        "/internal*" { pool service_group_internal }
        default { pool service_group_dynamic }
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

table

Description Provides a mechanism to store and retrieve key-value pairs in memory. Useful for caching, session tracking, or custom logic flows.

Syntax

```
table <operation> <table_name> <key> [value]
```

Valid Events

All

For information about aFleX events, see aFleX Events.

For information about table commands, see Table Commands.

use

Description This command is used to deal with backwards compatibility. It must be paired with the `node`, `pool` or other ACOS command. It is recommended that use of these commands be done directly rather than the `use` command.

Syntax

```
use <object> <object_name>
```

Example

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
        use pool service_group_static
    } elseif { [HTTP::uri] ends_with ".asp" } {
        use pool service_group_dynamic
    }
}
```

Example The use of the following example script is recommended instead.

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
        pool service_group_static
    } elseif { [HTTP::uri] ends_with ".asp" } {
        pool service_group_dynamic
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

utc_to_numeric_date

Description Converts a time value in UTC format (used in SAML tokens) into a NumericDate format, which is commonly used in JWT (JSON Web Token).

Syntax

```
utc_to_numeric_date [utc_time_string]
```

Example Use the following example to convert a UTC timestamp into JWT-compatible NumericDate format and log it:

```
when CLIENT_ACCEPTED {
    set utc_time "2025-04-29T14:33:00Z"
    set numeric_time [UTCtoNumericDate $utc_time]
    log "NumericDate format: $numeric_time"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

virtual

Description This command returns the name of the associated virtual server that the connection is flowing through.

Syntax

```
virtual name
```

Example

Use the following example to log the name of the virtual server handling the current HTTP request:

```
when HTTP_REQUEST {
    log "Virtual Server: [virtual name]"
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

when

Description

This command allows one to specify an event in an aFleX script. All aFleX *events* begin with a `when` command. Multiple `when` commands can be specified within a single aFleX script.

Syntax

```
when <event_name>
```

Example

The following example drops connections from a specific client IP address (192.168.1.10) when a client TCP connection is accepted.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10] } {
        drop
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

whereis

Description This command will return the geo-location information for a given IP address. The command will search in the geo-location database in use on the ACOS device. This can be helpful when used in a script that looks up information in a geo-location database from a third-party vendor.

Syntax

```
whereis <ipaddr>
```

Example The use of the following example takes a geo-location database from a third-party vendor to look up the location of clients who send requests to a specific VIP.

To use the geo-location database, it must be imported onto the ACOS device as a comma-separated values (.csv) file. To activate the database, it must be loaded. After loading the database, a template must be applied to specify the data fields so that the information can be extracted from the database.

The following example shows the use of a geo-location database that contains entries in the following format:

```
"0000000000","0016777215","","","",""
"0016777216","0016777471","AU","AUSTRALIA","OC","OCEANIA"
"0016777472","0016778239","CN","CHINA","AS","ASIA"
"0016778240","0016779263","AU","AUSTRALIA","OC","OCEANIA"
"0016779264","0016781311","CN","CHINA","AS","ASIA"
"0016781312","0016785407","JP","JAPAN","AS","ASIA"
"0016785408","0016793599","CN","CHINA","AS","ASIA"
"0016793600","0016809983","JP","JAPAN","AS","ASIA"
"0016809984","0016842751","TH","THAILAND","AS","ASIA"
...
"3758093312","3758094335","IN","INDIA","AS","ASIA"
"3758094336","3758095359","AU","AUSTRALIA","OC","OCEANIA"
"3758095360","3758095871","CN","CHINA","AS","ASIA"
"3758095872","3758096127","SG","SINGAPORE","AS","ASIA"
"3758096128","3758096383","AU","AUSTRALIA","OC","OCEANIA"
...
```

Each entry in this database has 6 fields. The aFleX script uses a GSLB CSV template to search the data in 4 of the fields:

```
"ip-from", "ip-to-mask", "country", "", "", "continent"
```

The use of the following example aFleX script performs search in the database:

```
when CLIENT_ACCEPTED {
    log "This is the country: [IP::client_addr]: [lindex
[whereis [IP::client_addr]] 0]"
    log "This is the continent: [IP::client_addr]: [lindex
[whereis [IP::client_addr]] 1]"
}
```

Example

The following steps illustrate the ACOS configuration steps required to install the geo-location database and use the aFleX script to search the data in the database.

- The following command will import a geo-location database file in .csv format onto the ACOS device:

```
ACOS#import geo-location ipligence-lite.csv use-mgmt-port
scp://root@192.168.209.80:/ipligence-lite.csv
Password []********
Importing ...
```

- The following commands will change the CLI to the global configuration level, and configure a template for extracting data from the geo-location database:

```
ACOS#config
ACOS(config)#gslb template csv geo-lookup
ACOS(config-gslb template csv)#field 1 ip-from
ACOS(config-gslb template csv)#field 2 ip-to-mask
ACOS(config-gslb template csv)#field 6 continent
ACOS(config-gslb template csv)#field 3 country
ACOS(config-gslb template csv)#exit
```

- The following commands will load the geo-location database (the .csv file) to activate it, and verify that it is loaded and activated:

```
ACOS(config)#gslb system geo-location load ipligence-lite.csv
geo-lookup
ACOS(config)#show gslb geo-location file
Per = Percentage of loading, Err/W = Error or Warning
T = T(Template)/B(Built-in)
```

```
Filename                  T Template                Per
Lines    Success  Err/W
-------------------------------------------------------------
----------------------
iana*                     B                         100% 77
    77       0
ipligence-lite.csv        T test                    100%
191805   191805   4
```

- The following command will text the database by searching the location information for a client IP address:

```
ACOS(config)#show gslb geo-location ip 74.125.224.33
                 Last = Last Matched Client, Hits = Count
of Client matched
                 T = Type, Sub = Count of Sub Geo-location
                     G(global)/P(policy), S(sub)/R(sub
range)
                     M(manually config)/B(built-in)

Global
Name           From            To              Last
   Hits     Sub  T
-------------------------------------------------------------
----------------------
NORTH AMERICA 74.124.206.88   74.126.95.255
   0        17821GR
.US
```

- The following command will add the aFleX script to the ACOS device. The script is copy-pasted into the CLI in this example. Alternatively, the script can be configured elsewhere and then imported as a file.

```
ACOS(config)#aflex create geo-lookup-script
Type in your aFleX script (type . on a line by itself when
done)
when CLIENT_ACCEPTED {
  log "Country=[lindex [whereis 74.125.224.35] 0]"
  log "Continent=[lindex [whereis 74.125.224.35] 1]"

  log "Country=[lindex [whereis 74.125.224.56] 0]"
  log "Continent=[lindex [whereis 74.125.224.56] 1]"
```

```
  log "Country=[lindex [whereis 123.125.114.144] 0]"
  log "Continent=[lindex [whereis 123.125.114.144] 1]"
}
.
aFleX geo-lookup-script created; syntax check passed.
```

- The following commands will bind the aFleX script to a virtual port.

```
ACOS(config)#slb virtual-server vip-L7-25-130
203.0.113.130
ACOS(config-slb vserver)#port 80 http
ACOS(config-slb vserver-vport)#aflex geo-lookup-script
ACOS(config-slb vserver-vport)#end
```

NOTE:

- The provided example does not show real servers and service group configuration, but they are required. In addition, network connectivity connection (Network Address Translation (NAT)), may also be needed.
- The end command is not a part of the VIP configuration. It functions by returning the CLI prompt to the Privileged EXEC configuration level.

After some traffic is sent to the VIP, the ACOS log will list the geo-location information for the client:

```
ACOS#show log
Log Buffer: 30000
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Continent=ASIA
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Country=CN
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Continent=NORTH
AMERICA
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Country=US
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Continent=NORTH
AMERICA
May 15 2012 04:15:37 Info    [AFLEX]:geo_test:Country=US
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

Global Variable Commands

You can use the following operators to quickly modify global variables across multiple parameters:

- array
- get
- incre
- set
- unset

For information about aFleX commands, see aFleX Commands.

For information about other global commands, see Global Commands.

array

Description This command will set or return elements in a global array.

NOTE: The maximum number of elements allowed in an array is 256. If the number of elements exceed 256, the command is rejected.

Syntax

```
array set <global_array> <key> <value>
```

This will set the values of one or more elements in the `<global_array>`.

```
array size <global_array>
```

This will extract the number of elements in the `<global_array>`.

```
array names <global_array>
```

This will provide a list of names for all the elements in the `<global_array>`.

```
array get <global_array> <key>
```

This will get a list of all pairs of elements in the `<global_array>`.

Valid Events

All.

For information about aFleX events, see aFleX Events.

get

Description This command will return the value of a global variable.

Syntax

```
get <global_variable>
```

Valid Events

All

For information about aFleX events, see aFleX Events.

incre

Description This command will increment the specified global variable by a value of 1. It is different from the Tcl command `incr` where it alters the global variables.

Syntax

```
incre <global_variable>
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

set

Description This command will set the value of a local variable.

Syntax

```
set <local_variable> <value>
```

This will set the `<local_variable>` to the specified `<value>`. When the variable does not exist, a new variable will be created upon this command's execution. The recommendation is to use `table set/delete` for global variables.

Valid Events

All.

For information about aFleX events, see aFleX Events.

unset

Description This command will unset the value of a local variable.

Syntax

```
unset <local_variable>
```

This will delete the value for the `<local_variable>`. It also forces the specified variable to return an empty string.

Valid Events

All.

For information about aFleX events, see aFleX Events.

AAM Commands

The following Application Access Management (AAM) commands are supported:

- AAM::attribute
- AAM::attribute_collection
- AAM::authentication
- AAM::authorization
- AAM::bypass
- AAM::client
- AAM::relay
- AAM::saml
- AAM::session

For information about aFleX commands, see aFleX Commands.

For information about AAM events, see AAM Events.

For additional information about AAM features, see the Application Access Management Guide.

AAM::attribute

Description This command will find the attribute value from an attribute collection in the AAM session according to the attribute name. This command will return correspondent attribute value to specific attribute name and multi-valued index.

There are a maximum of 16 attributes per collection.

Syntax

```
AAM::attribute get <attribute-name>
```

This returns the value of the attribute specified. Since neither a multi-valued index or collection ID are specified, the default value of 1 is assumed for both arguments.

By default, the attribute is returned as a string, but the type of data returned depends on the real authentication server's configuration:

- If an attribute on the server is an IP address or a string, this command will return string.
- If the attribute on the server is an integer, this command will return an integer.
- If there is no attribute with the specified name on the authentication server, this command returns an empty string.

```
AAM::attribute get <attribute-name> multi_value_id <num>]
```

This returns the value of the attribute specified, with the specified multi-valued index.

```
AAM::attribute get <attribute-name> collection-id <collection-id>
```

This returns the value of the attribute specified within the specified collection.

```
AAM::attribute get_multivalue_count <attribute-name>
```

This returns the number of values associated with the attribute specified. It returns 0 if the attribute doesn’t exist in the current auth session. To get the multi-value-count of attribute <attribute-name>,

you need to specify **AAM::attribute get <attribute-name>** before using **AAM::attribute get_multivalue_count <attribute-name>**.

NOTE: This command is only supported on HTTP and HTTPS virtual ports.

Example See Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

AAM::attribute_collection

Description This will specify a collection to be used by `<collection-id>`. If the backend authentication server type is LDAP, the ACOS device will only query the attributes defined in this collection.

There are a maximum of 16 collections per virtual port. The collection ID can be a number from 1-16. There can be a maximum of 16 attributes per collection.

Default If a collection ID is not specified during an `AAM_AUTHENTICATION_INIT` event, the ACOS device will use collection ID 1 as the default.

NOTE: This command is valid on HTTP and HTTPS virtual ports.

Syntax

```
AAM::attribute_collection <attribute-collection-id>
```

Example See Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT

AAM::authentication

Description This command will set or return authentication information.

Syntax

```
AAM::authentication get username
```

This will return the username for the authentication.

```
AAM::authentication set username <value>
```

This will set the username for the authentication to the specified value.

```
AAM::authentication get password
```

This will return the password for the authentication.

```
AAM::authentication set password <value>
```

This will set the password for the authentication to the specified value.

```
AAM::authentication get ntlm_domain
```

This will return the NTLM domain for the authentication.

```
AAM::authentication set ntlm_domain <value>
```

This will set the NTLM domain for the authentication to the specified value.

The `username`, `ntlm_domain` and `password` arguments are not supported for SAML authentication.

If NTLM logon is configured, the `ntlm_domain` and `username` arguments are only supported with the `get` option, and the `password` argument is not supported.

For RSA authentication servers, only `get username`, `set username` and `get password` are supported.

```
AAM::authentication set server <server_name>
```

This will set an authentication server to the specified name.

```
AAM::authentication set service-group <sg-name>
```

This will set an authentication service group to the specified name. The following are the return values:

0: Success

-1: No such server or service-group

-2: Wrong server or service-group type for this connection

-3: Internal system error

```
AAM::authentication get server-name
```

This will return the current authentication server or service group. If there is any AAM::authentication set server or service-group command executed before using this get command, you get the server or service-group configured by the previous command.

```
AAM::authentication get is-using-server
```

The following are the return values:

1: The current connection is using a server

0: The current connection is using a service group

-1: The current connection has no authentication.

Example Use the following example to append a different prefix to the username for authentication. The ACOS device will use the username `AUTH_$name` for authentication.

```
when AAM_AUTHENTICATION_INIT {
   set name "AUTH_"
   append name [AAM::client get username]
   AAM::authentication set username $name
```

Here is an example about set and get authentication and authorization configuration.

```
}when AAM_AUTHENTICATION_INIT  {
        regexp {^([^-]+)-(.+)$} [AAM::client get username] match user
domain

        log "=$user@$domain="
```

```
        AAM::authentication set username $user

        set ns1 [AAM::authentication get server-name]
        set zs1 [AAM::authorization get server-name]
        log "<CONF>  authn $ns1, authz $zs1"

        if { $domain equals "GROUP_100" } {
                AAM::authentication set server LDAP-198
                AAM::authorization set service-group ASG-LDAP
        }

        set ns2 [AAM::authentication get server-name]
        set zs2 [AAM::authorization get server-name]
        log "<AFLEX> authn $ns2, authz $zs2"
}
```

Example For additional examples, see Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT

AAM::authorization

Description This command will set or return authorization information.

Syntax

```
AAM::authorization set server <server-name>
```

This will set the authorization server to the specified value.

```
AAM::authorization set service-group <sg-name>
```

This will set the authorization service-group to the specified value.

```
AAM::authorization set ldap-search-filter <search-filter>
[disable-ldap-base-dn-from-cert]
```

This will set the LDAP search filter to the specified value.The maximum length of the search filer is 255. Disable 'use Subject DN as LDAP search base DN' using the disable option. This option is for the client-SSL template only. The following are the return values:

0: Success

-1: No such server or service-group

-2: Wrong server or service-group type for this connection

-3: Internal system error

```
AAM::authorization get server-name
```

This will return the current authorization server or service-group name. If there is any AAM::authorization set server or service-group command executed before using this get command, you get the server or service-group configured by the previous command.

```
AAM::authorization get is-using-server
```

The following are the return values:

1: The current connection is using a server

0: The current connection is using a service group

-1: The current connection has no authorization.

For additional examples, see Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT

AAM::bypass

Description This command will skip the authentication of a specific real server destination port through EP.

Syntax

```
AAM::bypass
```

Example Use the following example for specific real server destination port.

```
when HTTP_REQUEST {
if {[HTTP::host} contains ":1433" } {
AAM::bypass
}
}
```

Valid Events

- HTTP_REQUEST

AAM::client

Description This command will return information about user input.

Syntax

```
AAM::client get username
```

This will return the username that was input by the user.

```
AAM::client get password
```

This will return the password that was input by the user.

```
AAM::client get ntlm_domain
```

This will return the NTLM domain that was input by the user. For example, if the input for the username were in the format “domain\username”, this command would return the string “domain”.

```
AAM::client get authn_realm
```

This will return the realm used for authentication. For example, if the input for username were in the format “domain\username” or “username@domain”, this command would return the string “domain”.

Example Use the following example to append a different prefix to the username for authentication and relay. The ACOS device will use the username `AUTH_$name` for authentication, `RELAY_$name` for relay, and `$name` for authorization.

```
when AAM_AUTHENTICATION_INIT {
        set name [AAM::client get username]
```

```
    AAM::authentication set username "AUTH_$name" AAM::relay
set username "RELAY_$name"
}
```

Example For additional examples, see Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

AAM::relay

Description This will set or return information about relay.

Syntax

```
AAM::relay get username
```

This will return the username used in relay.

```
AAM::relay set username <value>
```

This will set the username used in relay to the specified value.

```
AAM::relay get password
```

This will return the password used in relay.

```
AAM::relay set password <value>
```

This will set the password used in relay to the specified value.

```
AAM::relay get realm
```

This will return the realm used in relay.

Example Use the following example to append a different prefix to the username for authentication and relay. The ACOS device will use the username `AUTH_$name` for authentication, `RELAY_$name` for relay, and `$name` for authorization.

```
when AAM_AUTHENTICATION_INIT {
        set name [AAM::client get username]
    AAM::authentication set username "AUTH_$name"
        AAM::relay set username "RELAY_$name"
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT

AAM::saml

Description SAML is an XML-based markup language for security assertions. This command returns information about the XML elements.

Syntax

```
AAM::saml get <path>
```

This command parses the SAML Assertion XML entity and gets XML element content or attributes from it. The path variable is the XML element tree path for the SAML Assertion XML entity.

Example The following is an example:

```
<saml:Assertion>
    <saml:Subject>
        <saml:NameID Format="unspecified"> test </saml:NameID>
    </saml:Subject>
</saml:Assertion>
```

Get content: To get the content of NameID element in the XML entity use: "AAM::saml get Assertion.Subject.NameID". The result is "test" in this case.

Get attribute: To get the Format attribute value of NameID element in the XML entity use: "AAM::saml get Format@Assertion.Subject.NameID". The result is "unspecified" in this case.

Syntax

```
AAM::saml get_multivalue_count <path>
```

This command parses the SAML Assertion XML entity and gets the number of contents for an XML element. The path variable is the XML element tree path for the SAML Assertion XML entity.

Example

To get the number of content elements for NameID element in the XML entity, use "AAM::saml get_multivalue_count Assertion.Subject.NameID". The result is 2 in this case.

```
<saml:Assertion>
    <saml:Subject>
        <saml:NameID Format="string"> test1 </saml:NameID>
        <saml:NameID Format="string"> test2 </saml:NameID>
    </saml:Subject>
</saml:Assertion>
```

Example

For more examples with SAML, see Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

AAM::session

Description This will return information about the auth session.

Syntax

```
AAM::session get matched_aaa_policy
```

This will return the name of the AAA policy matched for this session. This command can be used before the authentication session is established.

```
AAM::session get matched_aaa_rule
```

This will return the AAA rule index this session matched. This command can be used before the authentication session is established.

```
AAM::session get cookie_domain
```

This will return the cookie domain of this session. This command must be used after the authentication session is established. If it is used before the authentication session is established, it will return an empty string.

```
AAM::session get cookie_domain_group
```

This will return the cookie domain groups of this session. This command must be used after the authentication session is established. If it is used before the authentication session is established, it will return an empty string.

```
AAM::session set jwt <jwt-message>
```

This will set the constructed JWT message to the session, so for the next client request there is no need to re-construct it and it can be received through the "AAM::session get jwt" command directly.

Example The following is an example:

```
AAM::session set jwt "$jwt_msg"
```

Syntax

```
AAM::session get jwt
```

This will get the constructed JWT from the session if it is set through the "AAM::session set jwt <jwt-message>" command before.

Example The following is an example:

```
set jwt_msg [AAM::session get jwt]
```

Example See Example 7: Getting a constructed JWT from a Session.

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

Example AAM aFleX Scripts

This section contains the following examples of how to use aFleX scripts for authentication and authorization:

Example 1: Processing aFlex Commands in AAM_AUTHORIZATION_CHECK Event

Example 2: Classifying AAA Policy Result while Authenticating and Authorizing

Example 3: Setting Authentication Service-group by Requested Domain

Example 4: Setting Authorization Server by Client IP Address

Example 5: Selecting Domain-based Auth Server

Example 6: Get Scripts for Domain-based Auth Server Selection

Example 7: Getting a constructed JWT from a Session

Example 1: Processing aFlex Commands in AAM_AUTHORIZATION_CHECK Event

NOTE: When the command `reject` is used during the `AAM_AUTHORIZATION_CHECK` event, the authentication session is not established.

```
when AAM_AUTHENTICATION_INIT {
        if { [IP::addr [IP::client_addr] equals 192.168.1.0/16] } {
                AAM::attribute_collection 1
        } else {
                AAM::attribute_collection 2
        }
}

when AAM_AUTHORIZATION_CHECK {
        if { [IP::addr [IP::client_addr] equals 192.168.1.0/16] } {
                set username_c1 [AAM::attribute get UserName collection_id 1]

                if { $username_c1 ends_with "BadUser"} {
                        reject
                }
        } else {
                set username_c2 [AAM::attribute get UserName collection_id 2]
                set title [AAM::attribute get title collection_id 2]
                set business_unit [AAM::attribute get businessCategory collection_id 2

                for { set i 1} { $i <= [AAM::attribute get_multivalue_count
objectClass]} {incr i} {
                        set obj$i [AAM::attribute get objectClass multi_value_id $i co
id 2]
                }

                if { $username_c2 ends_with "spam"} {
                        reject
                } elseif {$title contains "sales"} {
                        reject
                } elseif {$business_unit starts_with "bad" } {
                        reject
                } elseif {$obj1 starts_with "inet" and $obj2 ends_with "Auth" and $obj
matches {p*n}} {
                        reject
```

```
                }
        }
}
```

Example 2: Classifying AAA Policy Result while Authenticating and Authorizing

```
when AAM_AUTHENTICATION_INIT {
        if { [AAM::session get matched_aaa_rule] equals 3 and [AAM::session get
matched_aaa_policy] equals "ldap"}
                AAM::attribute_collection 2
        } else {
                AAM::attribute_collection 1
        }
}

when AAM_AUTHORIZATION_CHECK {
        if { [AAM::session get matched_aaa_rule] equals 3 and [AAM::session get
matched_aaa_policy] equals "ldap"} {
                set username [AAM::attribute get UserName collection_id 2]

                if { $username ends_with "BadUser"} {
                        reject
                }

        } else {
                set username_c2 [AAM::attribute get displayName collection_id 1]
                set businessCategory [AAM::attribute get businessCategory collection_i
1]

                if { $username_c2 ends_with "ExampleName"} {
                        reject
                } elseif { $businessCategory starts_with "bad@" } {
                        reject
                }
        }
}
```

Example 3: Setting Authentication Service-group by Requested Domain

```
when HTTP_REQUEST {
```

```
                set reqhost [HTTP::host]
        }

        when AAM_AUTHNTICATION_INIT {
                if { $reqhost equals “secure.example.domain.com” } {
                        AAM::authentication set service-group “SECURE-LDAP-GROUP”
                }
                # use authenticaion server/service-group in configuration
        }
```

Example 4: Setting Authorization Server by Client IP Address

```
when AAM_AUTHENTICATION_INIT {
        if { [IP::addr [IP::client_addr] equals 198.168.0.0] } {
               AAM::authorization set server “LDAP-INTERNAL”
        } else {
               AAM::authorization set server “LDAP-EXTERNAL”
        }
}
```

Example 5: Selecting Domain-based Auth Server

There may be websites that users from multiple domains access. For example, these users can belong to the same parent company but different departments. Each of the different deparments have their own domains with their own identity store. The domains are not trusted. The user is required to enter the "domain/realm" with the username. The domain or realm may either be a separate field from the username, or concatenated with the username. In such a case, use the domain or realm to decide which authentication and authorization servers to choose, because their backend identity store and access control requirements may be different. For example, all the users in BU_1 domain are authenticated and authorized by BU_1 servers and those in BU_2 will go to BU_2 servers.

The following section discusses some scenarios.

Domain and Username as domain/user or user@domain

```
        when AAM_AUTHENTICATION_INIT {
                # Get domain
```

```
                set user_domain [AAM::client get authn_realm]

                if { $user_domain equals "BU_1" } {
                        AAM::authentication set server "AUTHN_SVR_BU_1"
                   AAM::authorization set server "AUTHZ_SVR_BU_1"
                } else if { $user_domain equals "BU_2" }
                        AAM::authentication set server "AUTHN_BU_2"
                        AAM::authorization set server "AUTHZ_BU_2"
                }
```

Domain Concatenated with Username

```
        when AAM_AUTHENTICATION_INIT {
                # Get domain from [domain-username]
                regexp {^([^-]+)-(.+)$} [AAM::client get username] match user_
domain user_name

                if { $user_domain equals "BU_1" } {
                        AAM::authentication set server "AUTHN_SVR_BU_1"
                   AAM::authorization set server "AUTHZ_SVR_BU_1"
                } else if { $user_domain equals "BU_2" }
                        AAM::authentication set server "AUTHN_BU_2"
                        AAM::authorization set server "AUTHZ_BU_2"
                }
        }
```

Domain in Customized HTTP Header

```
        when AAM_AUTHENTICATION_INIT {
                # Get domain from customized header UserDomain
                set user_domain [HTTP::header UserDomain]

                if { $user_domain equals "BU_1" } {
                        AAM::authentication set server "AUTHN_SVR_BU_1"
                   AAM::authorization set server "AUTHZ_SVR_BU_1"
                } else if { $user_domain equals "BU_2" }
                        AAM::authentication set server "AUTHN_BU_2"
                        AAM::authorization set server "AUTHZ_BU_2"
                }
        }
```

Example 6: Get Scripts for Domain-based Auth Server Selection

Before the set command:

```
        when AAM_AUTHENTICATION_INIT {
                set an1 [AAM::authentication get server-name]
                set az1 [AAM::authorization get server-name]
                log "configure: authenticate $an1, authorize $az1"
                # get the name of authentication and authorization from
configuration

                set user_domain [AAM::client get authn_realm]
                if { $user_domain equals "BU_1" } {
                        AAM::authentication set server "AUTHN_SVR_BU_1"
                        AAM::authorization set server "AUTHZ_SVR_BU_1"
                }
        }
```

After the set command:

```
        when AAM_AUTHORIZATION_CHECK {
                set an2 [AAM::authentication get server-name]
                set az2 [AAM::authorization get server-name]
                log "authenticated by $an2, authorized by $az2"
                # if user domain is BU_1, the variables should be AUTHN_SVR_BU_1
                # and AUTHZ_SVR_BU_1, else they will be the same with $an1 and
$az1
```

Server and service-group with same name:

```
        when AAM_AUTHORIZATION_CHECK {
                set an [AAM::authentication get server-name]
                set flag [AAM::authentication get is-using-server]
                if { 1 equals flag } {
                        log "authenticated by server: $an"
                } elseif { 0 equals flag } {
                        log "authenticated by service-group $an"
                } else { # -1
                        log "No authentication"
                }
        }
```

Example 7: Getting a constructed JWT from a Session

```
when AAM_AUTHENTICATION_INIT {
  AAM::attribute_collection 1
}

when HTTP_REQUEST_SEND {
  set jwt_msg [AAM::session get jwt]
  if { $jwt_msg equals "" } {
    # check necessary jwt contents
    set name [AAM::attribute get Fname collection_id 1]
    set role_count [AAM::attribute get_multivalue_count MemberOf]
    set nameId [AAM::saml get
Assertion.Conditions.AudienceRestriction.Audience]
    set nbf_str [AAM::saml get NotBefore@Assertion.Conditions]
    set exp_str [AAM::saml get NotOnOrAfter@Assertion.Conditions]
    set attr_cnt [AAM::saml get_multivalue_count
Assertion.AttributeStatement.Attribute]

      # hdr
      set jwt_hdr [b64encode "{ \"alg\": \"ES256\", \"typ\": \"JWT\"}"]
      log local0.0 "hdr = { \"alg\": \"ES256\", \"typ\": \"JWT\"}"

      # payload (from SAML attributes)
      set raw_payload "{ \"user\": \"$name\""

      set exp [utc_to_numeric_date $exp_str]
      set raw_payload "$raw_payload, \"exp\": $exp"

      set nbf [utc_to_numeric_date $nbf_str]
      set raw_payload "$raw_payload, \"nbf\": $nbf"

      for {set i 1} {$i <= $attr_cnt} {incr i} {
        set attr_name [AAM::saml get
Name@Assertion.AttributeStatement.Attribute.$i]
        set attr_val [AAM::saml get
Assertion.AttributeStatement.Attribute.$i.AttributeValue]
        set raw_payload "$raw_payload, \"$attr_name\": \"$attr_val\""
      }
      set raw_payload "$raw_payload}"
      set jwt_payload [b64encode $raw_payload]
```

```
      # signature
      set jwt_signature [b64encode [esha256 "$jwt_hdr.$jwt_payload" ec_
256]]

      # jwt
      set jwt_msg "$jwt_hdr.$jwt_payload.$jwt_signature"
      AAM::session set jwt $jwt_msg
  }

  if { not ($jwt_msg equals "") } {
    HTTP::header insert Authorization "Bearer $jwt_msg"
  }
}
```

AES Commands

The following Advanced Encryption Standard (AES) commands are supported:

- AES::decrypt
- AES::encrypt
- AES::key

For information about aFleX commands, see aFleX Commands.

AES::decrypt

Description This command will use an AES key to decrypt content.

Syntax

```
AES::decrypt <key> <content>
```

Example Use the following example to set the key and log a message about decrypted content.

```
when HTTP_REQUEST {
    set key [AES::key password 256]
    log "The AES decrypted content is [AES::decrypt $key
[HTTP::payload]]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

AES::encrypt

Description This command will use an AES key to encrypt the content.

Syntax

```
AES::encrypt <key> <content>
```

Example Use the following example to set the key and log a message about encrypted content.

```
when SERVER_DATA {
    set key [AES::key password 192]
    log "The AES encrypted content is [AES::encrypt $key
[TCP::payload]]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

AES::key

Description This command will use a randomly created key for encrypting/decrypting data using AES.

The key returned has the following format: `<8-byte-header><16-byte-IV><16/24/32-byte-key>`.

The 8-byte header is of the form “AES xxx” where xxx is 128, 192, or 256. The resulting key file can be 40, 48, or 56 bytes long.

Syntax

```
AES::key <passphrase> [256 | 192 | 128]
```

The `[256 | 192 | 128]` option specifies the key length, in bits. The default is 128.

Example Use the following example to log a message about the AES key.

```
when SERVER_DATA {
    log "The AES key is [AES::key password]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

Application Firewall Commands

The following commands related to application firewall are supported:

- APPCLS::application

For information about aFleX commands, see aFleX Commands.

APPCLS::application

Description Use the command to view information about the application protocol and category name from the connection. The command is only supported when you have configured application firewall for your ACOS system and you have a valid QOSMOS license. The command only supports TCP and UDP data plane events and TCP and UDP type services. At least one single application firewall rule must be configured to enable application classification. Application classification needs several packet exchanges, so you cannot predict at which aFleX event the classification is completed. Application Level Gateway (ALG) is not supported.

Syntax

```
APPCLS::application get protocol
```

Use the aforementioned command to return a list of classified application protocol names.

Syntax

```
APPCLS::application get classification-path
```

Use the aforementioned command to return the application classification path.

Syntax

```
APPCLS::application get category
```

Use the aforementioned command to return a list of application category names.

Example The following script returns a list of application protocol names. The value returned can be pending for a pending state, a blank string for no application protocol names, or the name of the application protocol if one is configured.

```
when HTTP_REQUEST {
  log "app protocol = '[APPCLS::application get protocol]'"
}
```

Example The following script returns a classification path. The value returned can be a blank string for no application classification path, or the name of the application classification path if one is configured.

```
when HTTP_REQUEST {
  log "app path = '[APPCLS::application get classification-
path]'"
}
```

Example The following script returns the list of application category names. The value returned can be pending for a pending state, a blank string for no category names, or the names of the categories of the most classified protocols.

```
when HTTP_REQUEST {
  log "app category = '[APPCLS::application get category]'"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERHELLO
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

Category Commands

The following category commands is supported:

- CATEGORY::lookup

For information about aFleX commands, see aFleX Commands.

CATEGORY::lookup

Description It accepts one parameter (URL) input. This returns the web category received from local library or Bright Cloud server.

Syntax

```
CATEGORY::lookup
```

NOTE: This command supports new proxy.

```
CATEGORY::lookup <url> [require-web-category]
```

NOTE: The `require-web-category` option is used to enable run-time-update. This option works with both HTTP/1.1 and HTTP/2 connections and is only applicable to HTTP_REQUEST and HTTP_REQUEST_DATA events.

Example An example for a web category lookup is mentioned below:

```
when HTTP_REQUEST {
        set uri [HTTP::uri]
        set cats [CATEGORY::lookup $uri]
        set i 1
        foreach cat $cats {
        log local0.0 "HTTP request: num: $i category: $cat"
        incr i
        }
}
```

An example for a web category cloud lookup is mentioned below:

```
when HTTP_REQUEST {
    set host [HTTP::host]
      set cat [CATEGORY::lookup $host require-web-category]
      log "host='$host' cat='$cat'"
}
```

An example for a web category cloud lookup is mentioned below:

```
when DNS_REQUEST {
log "Received DNS request for: [DNS::question name]"
set query_name [DNS::question name]

set cat [CATEGORY::lookup $query_name]
foreach cat $cats {
log  "HTTP request: num: category: $cat"
if {$cat == "search-engines"} {
   log "match"
}
}
DNS::return
}
```

An example of an asynchronous web category lookup during an HTTP_REQUEST event is mentioned below:

```
when HTTP_REQUEST {
set host [HTTP::host]
log "Category : [CATEGORY::lookup $host require-web-category]"
}
```

An example of an asynchronous web category lookup during an HTTP_REQUEST_DATA event is mentioned below:

```
when HTTP_REQUEST {
HTTP::collect
}
when HTTP_REQUEST_DATA {
set host [HTTP::host]
log "Category : [CATEGORY::lookup $host require-web-category]"
}
```

Valid Events

Valid with the following AAM events:

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT

Valid with the following HTTP events:

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

Valid with the following DNS events:

- DNS_REQUEST

Class List Commands

The following class list commands are supported, but currently limited to non-Aho-Corasick access lists:

- CLASS::exists
- CLASS::match
- CLASS::names
- CLASS::type

For information about aFleX commands, see aFleX Commands.

NOTE: The class-list must be configured and attached to the same vport as the aFleX script using a policy template.

NOTE: Class list commands require the LID to be defined in the configuration, either globally or on the virtual-server or virtual port.

NOTE: Multiple LID definitions may be available for a non-global LID. This includes a LID in a policy template bound to a virtual port, a LID in a DNS template bound to a virtual port, a LID in a policy template bound to a virtual server, and a LID configured in a system-wide policy template. For more information, see Limit ID Commands.

CLASS::exists

Description This example will return a Boolean value that indicates whether the class list exists.

Syntax

```
CLASS::exists <list-name>
```

Example Use the following example to log when a class list exists.

```
when HTTP_REQUEST {
    log "The class exists for [CLASS::exists example_list]."
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

CLASS::match

Description Queries class lists to check for matches and returns any component of a matching entry.

NOTE: Queries to a string class list are case sensitive. Queries to a DNS class are not case sensitive.

NOTE: In this release, string class lists can be referenced by name and externally modified.

NOTE: Class commands read class lists only and do not modify the entries in any way.

For Class List of Types Other than String

Syntax

```
CLASS::match <param> <list-name> [ip | dns]
```

This will return whether `<param>` matches an `[ip | dns]` entry in class list `<list-name>`. Omitting the `[ip | dns]` argument will result in IP entries in the class list being searched first, followed by DNS entries.

```
CLASS::match <param> <list-name> <key> [ip | dns]
```

This will return the key of the match when `<param>` matches an `[ip | dns]` entry in class list `<list-name>`. Omitting the `[ip | dns]` argument will result in IP entries in the class list being searched first, followed by DNS entries.

```
CLASS::match <param> <list-name> <lid> [ip | dns]
```

This will return the LID of the match (only if configured) when `<param>` matches an `[ip | dns]` entry in classlist `<list-name>`. Omitting the `[ip | dns]` argument will result in IP entries in the class list being searched first, followed by DNS entries.

Example Use the following example to log matches based on parameters.

```
when HTTP_REQUEST {
    log "CLASS Match: [CLASS::match 192.168.1.10 example_
list]"
    log "CLASS Match key: [CLASS::match 192.168.1.10 example_
list key]"
    log "CLASS Match lid: [CLASS::match 192.168.1.10 example_
list lid]"
}
```

NOTE: To perform a comparison of IP address 192.168.10.1 with a list of addresses in a class list that has been set as $classlist, use either of the following lines of syntax:

```
[CLASS::match 192.168.10.1 $classlist ip]
```

or

```
[CLASS::match [IP::client_addr] $classlist ip]
```

Use of `IP::addr` is not necessary if the `CLASS::match` command is used to perform address-to-address comparison.

For Class Lists of Type String

Syntax

```
CLASS::match <param> <operator> <list-name>
```

This will return whether `<param>` matches an entry in class list `<classname>`.

```
CLASS::match <param> <operator> <list-name> <key>
```

This will return the key of the match when `<param>` matches an entry in class list `<list-name>`.

```
CLASS::match <param> <operator> <list-name> <lid>
```

This will return the LID of the match when `<param>` matches an entry in class list `<list-name>`.

```
CLASS::match <param> <operator> <list-name> <value>
```

This will return the value of the match when `<param>` matches an entry in class list `<list-name>`.

The `<operator>` can be any of the following: `starts_with`, `ends_with`, `contains`, `equals`

NOTE: The maximum number of string entries for a class list depends on the total available system memory of the ACOS device.

- Memory Size 80GB or greater – 64K entries
- Memory Size 40GB or greater – 32K entries
- Memory Size 15GB or greater – 16K entries
- Memory Size 7GB or greater – 8K entries
- Memory Size 7GB or less – 4K entries

Example

```
when HTTP_REQUEST {
    log "The class match is [CLASS::match example.com ends_
with example_hosts]"
    log "The class match key is [CLASS::match www starts_with
example_hosts key]"
    log "The class match lid is [CLASS::match www.example.com
equals example_hosts lid]"
    log "CLASS Match value: [CLASS::match www.example.com
equals example_hosts value]"
}
```

Example Use the following example to redirect an HTTP request to the URL that has an entry in the class list.

```
when HTTP_REQUEST {
    set uri [string tolower [HTTP::uri]]
    set redirect_url [CLASS::match $uri equals value]
    if { not ($redirect_url equals "") } {
        HTTP::redirect $redirect_url
        log "The redirected $uri is $redirect_url" }
    }
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

CLASS::names

Description This command will return a list of class-list names.

Syntax

```
CLASS::names
```

Example Use the following example to log class-list names.

```
when HTTP_REQUEST {
    log "CLASS Name: [CLASS::names]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

CLASS::type

Description This command will return the type of the specified class list.

The type value that can be returned by aFleX depends on whether the type was explicitly specified during class-list configuration. If the type is a pair of empty brackets ([]), the class list does not contain any entries.

- Explicitly configured: dns, ipv4, ipv6, string
- Implicitly configured by the ACOS device based on the class-list entries: [], [dns], [ipv4], [ipv6], [dns, ipv4], [dns, ipv6]

Syntax

```
CLASS::type <list-name>
```

Example

Use the following example to log the class type for the class-list name.

```
when HTTP_REQUEST {
    log "The class type for example_ips is [CLASS::type
example_ips]"
    log "The class type for example_hosts is [CLASS::type
example_hosts]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED

- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Compression Commands

The following compression commands are supported on HTTP traffic (original proxy) and HTTP2 traffic (new proxy):

- COMPRESS::brotli
- COMPRESS::disable
- COMPRESS::enable
- COMPRESS::gzip

For information about aFleX commands, see aFleX Commands.

COMPRESS::brotli

Description Brotli (RFC 7932) is a lossless compression technique that compresses data utilizing a combination of the LZ77 algorithm. ADC supports Brotli compression and decompression for HTTP/2 protocol and HTTP/1 traffic is also supported when compression algorithm is specified through method order command under `http template` or via aFleX.

Syntax

```
COMPRESS::brotli level <level>
```

Specify the level at which the Brotli will be used.

```
COMPRESS::brotli sliding-window <sliding-window-size>
```

Specify the value of the window size (i.e. value of lgwin) of Brotli.

Example Use the following example to set the brotli compression level.

```
when HTTP_REQUEST {
COMPRESS::enable
COMPRESS::brotli level 4
}
```

Use the following example to set the brotli compression sliding window size.

```
when HTTP_REQUEST {
```

```
COMPRESS::enable
COMPRESS::brotli level 4
COMPRESS::brotli sliding-window 5
}
```

Use the following example to the order of the compression algorithm.

```
when HTTP_REQUEST {
COMPRESS::enable
COMPRESS::brotli level 4
COMPRESS::brotli sliding-window 5
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

COMPRESS::disable

Description This command will disable the compression for an HTTP response.

Syntax

```
COMPRESS::disable
```

Example Use the following example to check if a particular header response does not exist, and then disable compression.

```
when HTTP_RESPONSE {
    if { not ([HTTP::header exists "Accept-Encoding"]) } {
        COMPRESS::disable
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

COMPRESS::enable

Description This command will enable compression for an HTTP response.

NOTE: Only supports HTTP_REQUEST event on the HTTP2 (new proxy).

Syntax

```
COMPRESS::enable
```

Example

Use the following example to check if a particular header response exists, and if the uri ends with html, then enable compression.

```
when HTTP_RESPONSE {
    if { [HTTP::header exists "Accept-Encoding"] } {
        if { [HTTP::uri] ends_with ".html" } {
            COMPRESS::enable
        }
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

COMPRESS::gzip

Description

This command will set the level for HTTP compression.

Syntax

```
COMPRESS::gzip level <level>
```

The `<level>` can be 1-9.

NOTE: Setting the compression level to a higher value results in more HTTP compression at a greater CPU cost. Additional CPU usage can outweigh the benefit of a higher level. For example, setting compression to level 6 can provide equivalent performance to level 9. For best performance, A10 Networks recommends setting compression to level 1.

Only supports HTTP_REQUEST event on the HTTP2 (new proxy).

Example Use the following example to check if a particular header response exists, and if the uri ends with html, then set the level of compression and enable compress.

```
when HTTP_RESPONSE {
    if { [HTTP::header exists "Accept-Encoding"] } {
        if { [HTTP::uri] ends_with ".html" } {
            COMPRESS::gzip level 9
            COMPRESS::enable
        }
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

COMPRESS::method_order

Description Sets the order of the compression algorithm to use when multiple algorithms are available. This allows control over whether Brotli, GZIP, or other supported methods are prioritized when compressing responses. This command must be used after enabling compression via `COMPRESS::enable`.

Syntax

```
COMPRESS::method_order <method-order>
```

This command allows you to specify the priority of Brotli, GZIP, or other supported compression algorithms when compressing responses. It should be used only after enabling compression with `COMPRESS::enable`.

Example Use the following example to define the preferred order of compression algorithm:

```
when HTTP_REQUEST {
COMPRESS::enable
COMPRESS::brotli level 4
COMPRESS::brotli sliding-window 5
COMPRESS::method_order gzip brotli
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED

- SERVER_DATA

Configuration Commands

The following command related to configuration is supported:

- CONFIG::get

For information about aFleX commands, see aFleX Commands.

CONFIG::get

Description Retrieves configuration value by specified aXAPI object field name. Every field name is allowed, and the return can be an integer or string type.

Syntax

```
CONFIG::get <api-obj-url:field-name>
```

Example Use the following example to get the connection limit configuration by axAPI object URI for the SLB virtual server named vs1 and set the value to a local variable. If the variable value is greater than 100, then the example_service_group1 pool will be used. Otherwise, the example_service_group2 pool will be used.

```
when CLIENT_ACCECPTED {
    set limit [CONFIG::get "slb/virtual-server/vs1:conn-
limit"]
    if { $limit > 100 } {
        pool example_service_group1
    } else {
        pool example_service_group2
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

Database Load-Balancing Commands

The following commands related to database load balancing (DBLB) are supported:

- DB::command
- DB::query

For information about aFleX commands, see aFleX Commands.

For information about DBLB events, see Database Load-Balancing Events.

DB::command

Description The command returns a numeric value that represents the command number.

Syntax

```
DB::command
```

Example Use the following example to log the DB command value to the assigned service group.

```
when DB_COMMAND {
    log "DB Command: [DB::command]"
    pool mssg1_service_group
}
```

Valid Events

DB_COMMAND

DB::query

Description This command returns a string that holds the entire SQL query which was sent by the client.

Syntax

```
DB::query
```

Example Use the following example to log the DB query value to the assigned service group.

```
when DB_QUERY {
    log "DB Query: [DB::query]"
    pool mssg1_service_group
}
```

Valid Events

DB_QUERY

Diameter Load-Balancing Commands

You can use the following operators to quickly modify global variables across multiple parameters:

- DIAMETER::app_id
- DIAMETER::avp
- DIAMETER::cmd_code
- DIAMETER::length
- DIAMETER::version

For information about aFleX commands, see aFleX Commands.

For information about diameter load-balancing events, see Diameter Load-Balancing Events.

DIAMETER::app_id

Description This command returns the application ID of a Diameter message.

Syntax

```
DIAMETER::app_id
```

Example Use the following example to log the Diameter App ID value.

```
when DIAMETER_REQUEST {
        log "The DIAMETER::app_id is [DIAMETER::app_id]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND
- SERVER_CLOSED

DIAMETER::avp

Description This command is used to read, write, or delete AVPs.

Syntax

```
DIAMETER::avp count
```

This command returns the number of AVPs.

```
DIAMETER::avp get_ids [<avp_code> | <name>]
```

This command returns a list of the IDs of AVPs with matching `<avp_code>` or `<name>`. If the `<avp_code>` or `<name>` is not specified, the IDs of all AVPs are returned.

NOTE: The order of IDs might not be the same as the order of the AVPs in the packet.

```
DIAMETER::avp <id> code [name | type]
```

This command returns the numeric AVP code of the AVP with ID `<id>`. If the `[name]` is specified and the AVP is a standard AVP, a user-readable string is returned; otherwise, an empty string is returned. If `[type]` is specified, and the AVP is a standard AVP, its type is returned; otherwise, an empty string is returned.

```
DIAMETER::avp <id> index
```

This command returns the index value within the packet of the AVP with ID `<id>`.

```
DIAMETER::avp <id> flags
```

This command returns flags of the AVP with ID `<id>` in the following format: `{V|-}{M|-}{P|-}`

```
DIAMETER::avp <id> length
```

This command returns the length of the AVP with ID `<id>`.

```
DIAMETER::avp <id> vendor_id
```

This command returns the vendor_id of the AVP with ID `<id>` if the AVP has the “V” flag specified; otherwise, an empty string is returned.

```
DIAMETER::avp <id> value [<type>]
```

This command returns the value of the AVP with ID `<id>`. If the specified `<type>` is Unsigned32, Unsigned64, Integer32, Integer64, Address, or OctetString, the value is interpreted accordingly if it does not conflict with the AVP (for example, for an Integer32 AVP, Unsigned64 cannot be returned). For AVPs of type DiameterIdentity, Grouped, Time, DiamURI, Enumerated, or UTF8String, a byte array is returned.

```
DIAMETER::avp <id> delete
```

This command deletes the AVP with ID `<id>`.

NOTE: Once an AVP is deleted, it cannot be accessed thereafter.

```
DIAMETER::avp [index] insert [<avp_code> | <name>] <value>
<flags> [<vendor_id>] [type <type>]
```

This command inserts an AVP with the specified attributes. If `[index]` is specified, the AVP is inserted at that position in the packet. The `[index]` value must be between 0 and number of AVPs in the packet. Further, a packet can contain a maximum of 64 AVPs at any stage. If `[index]` is not specified, the AVP is appended to the packet. This command also returns the ID of the inserted AVP. If the `<avp_code>` is non-standard, the value is inserted as OctetString.

```
DIAMETER::avp <id> replace [value <value> [type <type>]]
[flags <flags> [<vendor_id>]]
```

This command replaces the value or flags (and vendor_id if flags includes "V") or AVP at ID `<id>`. The `<type>` can only be one of the following: Unsigned32, Unsigned64, Integer32, Integer64, Address, or OctetString.

Example Use the following example to log the AVP count, ID values, ID values for code 257, and session IDs.

```
when DIAMETER_REQUEST {
        log "Number of AVPs = [DIAMETER::avp count]"
        log "Ids of all AVPs = [DIAMETER::avp get_ids]"
        log "Ids of AVPs of code 257 = [DIAMETER::avp get_ids 257]"
        log "Ids of Session-Id AVPs  = [DIAMETER::avp get_ids
Session-Id]"
}
```

Example Use the following example to incrementally log ID codes, code names, code types, index values, flag values, and message lengths.

```
when DIAMETER_REQUEST {
        set ids [DIAMETER::avp get_ids]
        for { set i 0 } { $i < [llength $ids] } { incr i } {
        set id [lindex $ids $i]
        log "DIAMETER::avp $id code = [DIAMETER::avp $id code]"
        log "DIAMETER::avp $id code name = [DIAMETER::avp $id code
name]"
        log "DIAMETER::avp $id code type = [DIAMETER::avp $id code
type]"
```

```
        log "DIAMETER::avp $id index = [DIAMETER::avp $id index]"
        log "DIAMETER::avp $id flags = [DIAMETER::avp $id flags]"
        log "DIAMETER::avp $id length = [DIAMETER::avp $id length]"
        log "DIAMETER::avp $id vendor_id = [DIAMETER::avp $id vendor_
id]"
        log "DIAMETER::avp $id value = [DIAMETER::avp $id value]"
        }
}
```

Example Use the following example to incrementally delete AVP IDs.

```
when DIAMETER_REQUEST {
        set ids [DIAMETER::avp get_ids]
        for { set i 0 } { $i < [llength $ids] } { incr i } {
                set id [lindex $ids $i]
                DIAMETER::avp $id delete
        }
}
```

Example Use the following example to insert a new AVP to the Diameter message and then log AVP code, code names, code types, index values, flag values, message length, and vendor IDs of the new AVP.

```
when DIAMETER_REQUEST_SEND {
set newid [DIAMETER::avp insert 12345 6789 VMP 567 type
Unsigned32]
log "DIAMETER::avp $newid code = [DIAMETER::avp $newid code]"
log "DIAMETER::avp $newid code name = [DIAMETER::avp $newid
code name]"
log "DIAMETER::avp $newid code type = [DIAMETER::avp $newid
code type]"
log "DIAMETER::avp $newid index = [DIAMETER::avp $newid
index]"
log "DIAMETER::avp $newid flags = [DIAMETER::avp $newid
flags]"
log "DIAMETER::avp $newid length = [DIAMETER::avp $newid
length]"
log "DIAMETER::avp $newid vendor_id = [DIAMETER::avp $newid
vendor_id]"
log "DIAMETER::avp $newid value Unsigned32 = [DIAMETER::avp
$newid value Unsigned32]"
```

```
}
```

Example Use the following example to add a new AVP with ID 12345 and then replace the values for Flag and Type. Log AVP index, flags, message length, and vendor ID.

```
when DIAMETER_REQUEST_SEND {
set newid [DIAMETER::avp 0 insert 12345 6789 VMP 567 type
Unsigned32]
DIAMETER::avp $newid replace value 12345 type Unsigned32 flags
VMP 567
log "DIAMETER::avp $newid index = [DIAMETER::avp $newid
index]"
log "DIAMETER::avp $newid flags = [DIAMETER::avp $newid
flags]"
log "DIAMETER::avp $newid length = [DIAMETER::avp $newid
length]"
log "DIAMETER::avp $newid vendor_id = [DIAMETER::avp $newid
vendor_id]"
log "DIAMETER::avp $newid value Unsigned32 = [DIAMETER::avp
$newid value Unsigned32]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND
- SERVER_CLOSED

DIAMETER::cmd_code

Description This command returns the command code, or its name of a Diameter message. If [name] is specified, an empty string or one of the following

is returned as appropriate: ASR, ASA, ACR, ACA, CER, CEA, DWR, DWA, DPR, DPA, RAR, RAA, STR, or STA.

Syntax

```
DIAMETER::cmd_code [name]
```

If you use the `[name]` option, the name is returned,. If you omit the `[name]` option, the command code is returned instead.

Example Use the following example to log the Diameter code value.

```
when DIAMETER_REQUEST {
        log "DIAMETER::cmd_code = [DIAMETER::cmd_code]"
}
```

Example Use the following example to log the Diameter code name.

```
when DIAMETER_REQUEST {
        log "DIAMETER::cmd_code name = [DIAMETER::cmd_code name]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND
- SERVER_CLOSED

DIAMETER::length

Description This command returns the length of a Diameter message.

Syntax

```
DIAMETER::length
```

Example Use the following example to log the Diameter message length.

```
when DIAMETER_REQUEST {
        log "DIAMETER::length = [DIAMETER::length]"
```

```
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND
- SERVER_CLOSED

DIAMETER::version

Description This command returns the version of a Diameter message.

Syntax

```
DIAMETER::version
```

Example Use the following example to log the Diameter version value.

```
when DIAMETER_REQUEST {
        log "DIAMETER::version = [DIAMETER::version]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- DIAMETER_ANSWER
- DIAMETER_ANSWER_SEND
- DIAMETER_REQUEST
- DIAMETER_REQUEST_SEND
- SERVER_CLOSED

DNS Commands

The following DNS commands are supported:

- DNS::additional
- DNS::answer
- DNS::authority
- DNS::cache
- DNS::class
- DNS::header
- DNS::is_dnssec
- DNS::len
- DNS::name
- DNS::opt
- DNS::query
- DNS::question
- DNS::rdata
- DNS::return
- DNS::rr
- DNS::ttl
- DNS::type

For information about aFleX commands, see aFleX Commands.

For information about DNS events, see DNS Events.

DNS::additional

Description This command returns, inserts, removes, or clears RRs from the Additional section. With no arguments, the command returns a Tcl list of RR objects. With an argument, the command inserts/removes RR Tcl objects in the Additional section or clears all RRs from the Additional section.

Syntax

```
DNS::additional [[insert | remove rr_obj] | clear]
```

Example Use the following example to insert RR Tcl objects in the Additional section.

```
when DNS_RESPONSE {
set rr [DNS::rr "ns1.example.com. 3600 IN A 192.0.2.1"]
DNS::additional insert $rr
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::answer

Description This command returns, inserts, removes, or clears RRs from the Answer section. With no arguments, this command returns a Tcl list of RR objects. With an argument, this command inserts or removes RR Tcl objects in the Answer section or clears all RRs from the Answer section.

Syntax

```
DNS::answer [[insert | remove rr_obj] | clear]
```

Example Use the following example to set RR objects for a DNS response.

```
when DNS_RESPONSE {
  set rr [DNS::rr example.tld 149 IN A 127.0.0.10]
  DNS::answer insert $rr
  log "rrs = '[DNS::answer]'"
}
```

Example Use the following example to remove SOA records from the Answer section.

```
when DNS_RESPONSE {
    set rr [DNS::rr example.com 149 IN A 127.0.0.10]
    DNS::answer insert $rr
    log "DNS Answer: [DNS::answer]"
}
```

Example Use the following example to remove one RR from the answer.

```
when DNS_RESPONSE {
        set rrs [DNS::answer]
        set i 0
        foreach rr $rrs {
                log "i = $i rr = '$rr'"
                incr i
        }
        set rr1 [lindex $rrs 0]
        log "remove rr1 = '$rr1'"
        DNS::answer remove $rr1
        set k 0
        foreach rr [DNS::answer] {
                log "k = $k rr = '$rr'"
                incr k
        }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::authority

Description This command returns, inserts, removes, or clears RRs from the Authority section. With no arguments, this command returns a Tcl list of RR objects. With an argument, this command returns inserts or removes RR Tcl objects in the Authority section or clears all RRs from the Authority section.

Syntax

```
DNS::authority [[insert | remove rr_obj] | clear]
```

Example Use the following example to remove all the authority records.

```
when DNS_RESPONSE {
        set rrs [DNS::answer]
        set i 0
        foreach rr $rrs {
                log " i = $i rr ='$rr'"
                incr i
        }
        set rrs2 [DNS::authority]
        set j 0
        foreach rr2 $rrs2 {
                log "j = $j rr2 = '$rr2'"
                incr j
        }
        DNS::authority clear
}
```

Example Use the following example to remove a single authority record if there is more than one authority record.

```
when DNS_RESPONSE {
        set rrs2 [DNS::authority]
        set rr2 [lindex $rrs2 1]
        DNS::authority remove $rr2
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::cache

Description This command controls the DNS cache access and update for the current DNS session.

Syntax

```
DNS::cache <enable | disable>
```

NOTE: This command enables or disables the DNS cache for the current DNS session.

NOTE: This command is only effective when global DNS cache or a DNS cache template is enabled.

```
DNS::cache update
```

NOTE: This command updates the DNS cache with content changed through aFleX.

Example Use the following example to bypass the cached response for a DNSSEC query.

```
when DNS_REQUEST {
  if {[DNS::is_dnssec]} {
    log "This is DNSSEC request!"
    DNS::cache disable
  }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::class

Description This command gets or sets the resource record class field (IN, CH, HS, and so on).

Syntax

```
DNS::class <rr_obj> [value]
```

Example Use the following example to insert a record for a DNS response.

```
when DNS_RESPONSE {
    set rr [DNS::rr example.com 149 IN A 127.0.0.10]
    set rr1 [DNS::class $rr HS]
    DNS::answer insert $rr1
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::header

Description This command gets or sets simple bits or byte fields. Return value is always an integer except for successful recognition of the rcode or opcode fields, where a string is returned.

The rcode can be one of the following:

- NOERROR
- FORMERR
- SERVFAIL
- NXDOMAIN
- NOTIMPL
- REFUSED
- YXDOMAIN
- YXRRSET
- NXRRSET
- NOTAUTH
- NOTZONE

The opcode can be one of the following:

- QUERY
- IQUERY
- STATUS
- NOTIFY
- UPDATE

Syntax

```
DNS::header <id | qr | opcode | aa | tc | rd | ra | ad | cd |
rcode> [value]
```

NOTE: This command returns a read-only value.

```
DNS::header <qdcount | ancount | nscount | arcount> [value]
```

Example Use the following example to log all questions and responses for DNS requests and responses:

```
when DNS_REQUEST {
        log "Client: [IP::client_addr] Question:[DNS::question name]
Type:[DNS::question type] Class:[DNS::question class]"
        set fqdn [DNS::question name]
}
when DNS_RESPONSE {
        log "Request: $fqdn Answer: [DNS::answer] Status:
[DNS::header rcode] Flags: RD [DNS::header rd] RA [DNS::header
ra]"
}
```

Example Use the following example to log all query opcodes.

```
when DNS_REQUEST {
log "query id [DNS::header id] qr: [DNS::header qr] opcode:
[DNS::header opcode]"
}
when DNS_RESPONSE {
log "qr: [DNS::header qr] rcode: [DNS::header rcode] ra:
[DNS::header ra]"
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::is_dnssec

Description This command checks for a DNSSEC query or reply. It returns 1 if true and 0 if false.

Syntax

```
DNS::is_dnssec
```

Example Use the following example to check for a DNSSEC query.

```
when DNS_REQUEST {
```

```
        if {[DNS::is_dnssec]} {
                log "This is DNSSEC request!"
        }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::last_act

Description This command manages unhandled DNS queries and control the server's response behavior.

Syntax

```
DNS::last_act <allow|drop|reject|hint|noerror>
```

Example Use the following example to drop DNS requests that are not handled by a DNS service.

```
when DNS_REQUEST {
    DNS::last_act drop
}
```

DNS::len

Description This command returns the DNS packet message length.

Syntax

```
DNS::len
```

Example Use the following example to log the packet length for a DNS request.

```
when DNS_REQUEST {
    log "DNS len: [DNS::len]"
}
```

Example Use the following example to log the packet length for a DNS response.

```
when DNS_RESPONSE {
    log "DNS len: [DNS::len]"
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::name

Description This command gets or sets the resource record name field (FQDN); for example, “www.example.com”.

Syntax

```
DNS::name <rr_obj> [value]
```

Example Use the following example to set the FQDN for a DNS response.

```
when DNS_RESPONSE {
    set rr [DNS::rr www1.example.com 149 IN A 127.0.0.10]
    set rr1 [DNS::name $rr "www2.example.com"]
    DNS::answer insert $rr1
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::opt

Description This command gets or sets the parameters of a DNS OPT record. If there is no OPT record in the DNS content, the return value is NULL for ‘get’ commands.

Syntax

```
DNS::opt do [value]
```

NOTE: This command gets or sets the DO value for DNSSEC in an OPT record.

```
DNS::opt udpsize [value]
```

NOTE: This command gets or sets the UDP size value in an OPT record.

```
DNS::opt rcode [value]
```

NOTE: This command gets or sets the extended RCODE value in an OPT record.

```
DNS::opt version [value]
```

NOTE: This command gets or sets the version in an OPT record.

Example Use the following example to log DNS opt record for DNS requests and responses.

```
when DNS_REQUEST {
    if { [DNS::is_dnssec] } {
        log "This is DNSSEC request!"
        log "DNS opt udpsize: [DNS::opt udpsize]"
    }
}
when DNS_RESPONSE {
    if { [DNS::opt do] } {
        DNS::opt udpsize 8196
    }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::ptype

Description This command sets the resource type of a DNS packet during processing.

Syntax

```
DNS::ptype
```

Example Use the following example to modify the DNS response based on the packet type:

```
when DNS_RESPONSE {
    if { [DNS::ptype] eq "NXDOMAIN" } {
        log "NXDOMAIN response for [DNS::name]"
    }
}
```

DNS::query

Description This command returns a Tcl list of RR Tcl objects lists, one for each section: Answer, Authority, and Additional.

Syntax

```
DNS::query <target> <name> <type> [dnssec]
```

NOTE: The `<target>` can be “dnsx”. The `<name>` is the fully qualified domain name (for example, “www.example.com”). The `<type>` specifies the record type (A, AAA, MX, NPTR, and so on). The `dnssec` option gets DNSSEC data.

Example Use the following example to return RR Tcl objects for a DNS response.

```
when DNS_RESPONSE {
    set rrtcl [DNS::query dnsx ns1.example.com SOA]
    foreach rrs $rrtcl {
        foreach rr $rrs {
            if { [DNS::type $rr] equals "SOA" } {
                DNS::additional insert $rr
            }
        }
    }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::question

Description This command gets or sets the question field value. A question RR has no rdata and only requests with qdcount == 1 are accepted. The return types for name, type, and class are all strings. Type returns/accepts any of the valid DNS types defined in the RFCs. The class returns/accepts IN, CH, and HS.

Syntax

```
DNS::question <name | type | class> [value]
```

Example Use the following example to set a question field name and object for a DNS request and response.

```
when DNS_REQUEST {
    if { [DNS::question name] contains "internal.example.com"
} {
        log "DNS Question name: [DNS::question name]"
        DNS::question name "internal.example.com"
    }
}
when DNS_RESPONSE {
    set rr_ext [DNS::rr external.example.com 300 IN A
192.168.0.80]
    set rr_int [DNS::rr internal.example.com 300 IN A
192.168.0.0]
    if { [DNS::question name] contains "internal.example.com"
} {
        log "Original response question name: [DNS::question
name]"
        DNS::answer insert $rr_int
    } elseif { [DNS::question name] contains
"external.example.com" } {
        DNS::answer insert $rr_ext
    }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::rdata

Description This command gets or sets the resource record rdata field.

Syntax

```
DNS::rdata <rr_obj> [value]
```

Example Use the following example to set a resource record object for a DNS request.

```
when DNS_RESPONSE {
```

```
    set rr [DNS::rr example.com 149 IN A 127.0.0.10]
    set rr2 [DNS::rdata $rr "192.168.0.0"]
    DNS::answer insert $rr2
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::return

Description This command skips all further processing after Tcl execution and sends the DNS packet in the opposite direction.

Syntax

```
DNS::return
```

NOTE: When responding to a DNS query in the event DNS_REQUEST, you must use DNS::return in order to prevent the response being overwritten by the real DNS server or by the GSLB function when running GSLB on ACOS."

Example Use the following example to set a resource record name and object for a DNS request.

```
when DNS_REQUEST {
    if { [DNS::question name] contains "a10.example.com" } {
        DNS::header qr 1
        DNS::header ra 1
        set name [DNS::question name]
        set rr1 [DNS::rr $name 0 IN CNAME
vip1.a10.example.com]
        DNS::answer insert $rr1
        DNS::return
    }
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::rr

Description This command creates a new resource record object with the specified attributes.

Syntax

```
DNS::rr <name> <ttl> <class> <type> <rdata>
```

NOTE: The <name> is the FQDN (for example, “www.example.com”). The <ttl> specifies time to live in seconds. The <class> specifies the DNS class (IN, CH, HS, and so on). The <type> specifies the record type (A, AAA, MX, NPTR, and so on). The <rdata> value depends on the type of RR. For example for an A record, the <rdata> will be an IP address (“X.X.X.X”).

Example Use the following example to set a resource record object for a DNS response.

```
when DNS_RESPONSE {
    set rr [DNS::rr www.example.com 149 IN A 127.0.0.10]
    log "DNS rr: $rr"
}
```

Example Use the following example to set a resource record name and object for a DNS response.

```
when DNS_RESPONSE {
    set name [DNS::question name]
    set rr [DNS::rr $name 0 IN CNAME vip1.a10.example.com]
    DNS::answer insert $rr
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::ttl

Description This command gets or sets the resource record TTL field.

Syntax

```
DNS::ttl <rr_obj> [value]
```

Example Use the following example to set resource record TTL field for a DNS response.

```
when DNS_RESPONSE {
    set rr [DNS::rr example.com 149 IN A 127.0.0.10]
    set rr1 [DNS::ttl $rr 200]
    DNS::answer insert $rr1
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

DNS::type

Description This command gets or sets the resource record type field (A, AAAA, MX, NPTR, etc.).

Syntax

```
DNS::type <rr_obj> [value]
```

Example Use the following example to set the resource record for a DNS response.

```
when DNS_RESPONSE {
    set rr [DNS::rr example.com 149 IN A 127.0.0.10]
    set rr1 [DNS::type $rr CNAME]
    DNS::answer insert $rr1
}
```

Valid Events

- DNS_REQUEST
- DNS_RESPONSE

Financial Information eXchange Commands

The following commands related to Financial Information eXchange (FIX) are supported:

- FIX::begin_string
- FIX::body_length
- FIX::msg_seq_num
- FIX::msg_type
- FIX::sender_compid
- FIX::sending_time
- FIX::target_compid

For information about aFleX commands, see aFleX Commands.

For information about FIX events, see Financial Information eXchange Events.

FIX::begin_string

Description This command returns the value of the BeginString tag. The BeginString tag identifies the beginning of a new FIX message and the FIX protocol version. It is always the first field in the message and is always unencrypted.

Syntax

```
FIX::begin_string
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the beginning string for a FIX request.

```
when FIX_REQUEST {
    log "FIX begin_string: [FIX::begin_string]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::body_length

Description This command returns the value of the BodyLength tag. The FIX BodyLength tag gives the message length in bytes, forward to the CheckSum field. It is always the second field in the FIX message and is always unencrypted.

Syntax

```
FIX::body_length
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the body length of a FIX request.

```
when FIX_REQUEST {
    log "FIX body_length: [FIX::body_length] bytes"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::msg_seq_num

Description This command returns the integer message sequence number. It is always a positive value.

Syntax

```
FIX::msg_seq_num
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the message sequence number of a FIX request.

```
when FIX_REQUEST {
    log "FIX msg_seq_num: [FIX::msg_seq_num]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::msg_type

Description This command returns the value of the MsgType tag. The MsgType tag defines the message type, which is a string that is one or two characters in length. It is always the third field in the message and is always unencrypted.

Syntax

```
FIX::msg_type
```

NOTE: A “U” as the first character in the MsgType field (examples: U, U2, and so on) indicates that the message format is privately defined between the sender and receiver.

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the message type of a FIX request.

```
when FIX_REQUEST {
    log "FIX msg_type: [FIX::msg_type]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::sender_compid

Description This command returns the value of the SenderCompID tag. The SenderCompID is an assigned string value used to identify the firm sending the FIX message.

Syntax

```
FIX::sender_compid
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the sender company ID of a FIX request.

```
when FIX_REQUEST {
    log  "FIX sender_compid: [FIX::sender_compid]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::sending_time

Description This command returns the value of the time of message transmission, always expressed in UTC time. The time is returned as a string in either of the following formats:

- Whole seconds – `YYYYMMDD-HH:MM:SS`
- Milliseconds – `YYYYMMDD-HH:MM:SS.sss`

The colons, dash, and period are required.

Syntax

```
FIX::sending_time
```

NOTE: This timestamp is part of the transport level as a field in the StandardHeader and does not represent the time of a related business transaction. A timestamp for the business transaction is conveyed with the tag 60 TransactTime.

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the timestamp of a FIX request.

```
when FIX_REQUEST {
    log "FIX sending_time: [FIX::sending_time]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

FIX::target_compid

Description This command returns the value of the TargetCompID tag. The TargetCompID is an assigned string value used to identify the firm receiving the FIX message.

Syntax

```
FIX::target_compid
```

NOTE: This event is only valid on TCP-proxy and FIX virtual ports.

Example Use the following example to log the target company ID of a FIX request:

```
when FIX_REQUEST {
    log "FIX target_compid: [FIX::target_compid]"
}
```

Valid Events

- FIX_REQUEST
- FIX_RESPONSE

HTTP Commands

The following HTTP commands are supported on HTTP traffic (original proxy) and HTTP2 traffic (new proxy):

- HTTP::close
- HTTP::collect
- HTTP::cookie
- HTTP::disable
- HTTP::fallback
- HTTP::header
- HTTP::host
- HTTP::is_keepalive
- HTTP::is_redirect
- HTTP::method
- HTTP::path
- HTTP::password
- HTTP::payload
- HTTP::query
- HTTP::redirect
- HTTP::release
- HTTP::request
- HTTP::retry
- HTTP::request_num
- HTTP::respond
- HTTP::status
- HTTP::stream
- HTTP::uri

- HTTP::username
- HTTP::version

For information about aFleX commands, see aFleX Commands.

For information about HTTP events, see HTTP Events.

HTTP::close

Description This command will insert a “Connection: close” header and close the HTTP connection.

Syntax

```
HTTP::close
```

Example Use the following examples to close the HTTP connection after sending a response.

```
when HTTP_REQUEST {
    if { not ([IP::addr [IP::client_addr] equals
192.168.1.0/24]) } {
        HTTP::close
    }
}
```

```
when ICAP_RESPONSE {
if { not ([IP::addr [IP::client_addr] equals 10.10.10.10/24])
} {
HTTP::close
}
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::collect

Description This command will collect the amount of data specified using the <length> argument. When the system collects the specified amount of

HTTP content data, the aFleX event `HTTP_REQUEST_DATA` or `HTTP_RESPONSE_DATA` may be triggered depending on the source of the data.

`HTTP::request` or `HTTP::payload <size>` commands can be used with HTTP::collect.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
HTTP::collect
```

This will collect data. It is important to note when the content length is dropped, as it may strand your connection.

```
HTTP::collect [<length>]
```

This will collect the amount of data that is specified with the <length> argument. Specifying a value larger than the actual length may strand your connection.

NOTE: If length 0 is specified, the `HTTP_RESPONSE_DATA` event is not triggered since no data is collected.

- When the `<length>` option is not applied, the ACOS device behaves as follows:
- When the packet has an HTTP Content-Length header, the ACOS device will collect as much data as specified by the header, up to 1.25 MB, the maximum allowable limit.
- When the packet does not have an HTTP Content-Length header, the ACOS device keeps collecting data until one of the following occurs:
- The collection of 1.25 MB of data (This is the maximum limit.)
- A zero-size chunk-encoded packet is obtained
- RST is obtained from the server
- FIN is obtained from the server
- Typically, a packet without a Content-Length header is a chunk-encoded packet.

NOTE:

- The ACOS device will buffer the entire payload before responding to the client, so when the object to be collected is huge, there may be a performance hit.
- If RAM caching is enabled, the `HTTP::collect` command is not supported.
- When the `HTTP::payload` replace command is used in the same aFleX policy as the `HTTP::collect` command:
- For packets not containing chunk-encoded data, the ACOS device replaces the collected data with the specified string.
- For chunk-encoded packets, the command de-chunks the packet first, removing the chunk header and assembling the packet. The ACOS device will then replace the content with the new string without re-chunking the payload. The packet received by the client will not be chunk-encoded.
- The `HTTP::payload replace` command supports only clear text replacement. If the server response is compressed (transfer-encoded, tar, gz, bz, and so on), it will not work correctly. Therefore, when `HTTP::collect` is used in an aFlex policy (also with event `HTTP_RESPONSE`), the “Accept-Encoding” header will be automatically removed from the Request.

Example

Use the following example to collect the amount of data specified using the length argument.

```
when HTTP_RESPONSE {
if { ([HTTP::status] == 200) and ([HTTP::header "Content-
Type"] contains "text") } {
    if { [HTTP::header exists Content-Length] } {
        HTTP::collect [HTTP::header Content-Length]
    } else {
        HTTP::collect
    }
}
```

Valid Events

- HTTP_REQUEST

- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::cookie

Description This command is used to query or manipulate cookies in HTTP requests and responses. It replaces the `http_cookie` command.

By default, ACOS operates according to RFC 2109 and RFC 2965. When an extension attribute or an unknown attribute is encountered, ACOS stops parsing the remaining response cookie header and forwards the response to client as received from the server.

When the "cookie-format rfc6265" is bound to a vport in the HTTP template, ACOS treats the incoming set-cookie strings that do not conform to the RFC 6265 standard as extension variables. If there are multiple such attributes, only the last variable is stored and the previous ones are discarded. When the extension is free format, `HTTP::cookie` only supports sanitize for standard attributes.

NOTE: For SameSite attributes[1], configure the "cookie-format rfc6265" in the HTTP template and bind it to the vport for parsing.

Syntax

```
HTTP::cookie names
```

This will return the names of all the cookies present in the HTTP header.

```
HTTP::cookie count
```

This will return the number of cookies present in the HTTP header.

```
HTTP::cookie [value] <name> [string]
```

1 The "SameSite" attribute is not in the predefined list of RFC 2109 and RFC 2965 and is available only in the draft-updates of RFC 6265.

This will set or get the cookie value of the given name in an HTTP request. Drop the keyword `value` from this command if the cookie name does not collide with any of the other commands.

```
HTTP::cookie encrypt <name> <pass phrase> ["128" | "192" |
"256"]
```

Encrypts the value for the given cookie using a key generated from the pass phrase.

```
HTTP::cookie decrypt <name> <pass phrase> ["128" | "192" |
"256"]
```

Decrypts the value for the given cookie using a key generated from the pass phrase.

```
HTTP::cookie version <name> [version]
```

This will set or get the version of the cookie.

```
HTTP::cookie path <name> [path]
```

This will set or get the cookie path.

```
HTTP::cookie domain <name> [domain]
```

This will set or get the cookie domain.

```
HTTP::cookie ports <name> [portlist]
```

This will set or get the cookie port lists for V2 cookies.

```
HTTP::cookie insert name <name> value <value> [path <path>]
[domain <domain>] [version <0 | 1 | 2>]
```

This will add or replace a cookie in an HTTP response. The default value for the version is 0.

```
HTTP::cookie remove <name>
```

This will remove a cookie.

```
HTTP::cookie sanitize [attribute]+
```

This will remove everything except the specified attributes from the cookie.

```
HTTP::cookie exists <name>
```

This will return a true value if the cookie exists.

```
HTTP::cookie maxage <name> [seconds]
```

This will set or get the max-age. Version 1 cookies and response messages are only affected by this.

```
HTTP::cookie expires <name> [seconds] [absolute | relative]
```

This will set or get the expires attribute. Version 0 cookies are only affected. If an absolute argument is specified, the seconds value will represent the number of seconds based from the UNIX epoch, which is January 1, 1970. The default number of seconds is relative, which is the number of seconds from the current time. It applies to response messages only.

```
HTTP::cookie comment <name> [comment]
```

This will set or get the cookie comment. Version 1 cookies and response messages are only affected by this.

```
HTTP::cookie secure <name> [enable | disable]
```

This will set or get the value of the secure attribute. Response messages are only affected by this.

```
HTTP::cookie commenturl <name> [commenturl]
```

This will set or get the comment URL. Version 2 cookies and response messages are only affected by this.

```
HTTP::cookie discard <name> [enable | disable]
```

This will set or get the value of the discard attribute. Version 2 cookies and response messages are only affected by this.

NOTE: If both `HTTP::cookie` and `HTTP::header` commands are used to modify the same header, then `HTTP::cookie` takes precedence.

Example The following example aFleX script adds HttpOnly to all cookies set by the server.

```
when HTTP_RESPONSE {
    set current_time [TIME::clock seconds]
    foreach cookie_name [HTTP::cookie names] {
        if { [HTTP::cookie exists "$cookie_name"] } {
```

```
            set new_cookie "$cookie_name=[HTTP::cookie value
"$cookie_name"]"
            if { [HTTP::cookie expires "$cookie_name"] >
$current_time } {
                set cookie_expires [clock format [HTTP::cookie
expires "$cookie_name"] -format {%a, %d %b %Y %H:%M:%S GMT} -
gmt 1]
                append new_cookie "; Expires=$cookie_expires"
}
            if { [HTTP::cookie domain "$cookie_name"] ne "" }
{
                append new_cookie "; Domain=[HTTP::cookie
domain "$cookie_name"]"
            }
            if { [HTTP::cookie path "$cookie_name"] ne "" } {
                append new_cookie "; Path=[HTTP::cookie path
"$cookie_name"]"
            }
            append new_cookie "; HttpOnly"
            HTTP::cookie remove "$cookie_name"
            HTTP::header insert Set-Cookie "$new_cookie"
        }
    }
}
```

Example

The following example closes the HTTP connection for ICAP responses if the client's IP address is not within the specified range (192.168.1.0/24).

```
when ICAP_RESPONSE {
if { not ([IP::addr [IP::client_addr] equals 192.168.1.0/24])
} {
HTTP::close
}
}
```

Valid Events

- HTTP_REQUEST
- HTTP_RESPONSE
- ICAP_RESPONSE

HTTP::disable

Description This command will change an HTTP proxy from full parsing to pass-through mode.

When the command is used in the `HTTP_REQUEST` event, it disables the HTTP/HTTPS proxy, and traffic after that point will be processed by generic TCP proxy, or generic SSL proxy.

When the command is used in `HTTP_RESPONSE` event, it bypasses any HTTP related process for response traffic.

Syntax

```
HTTP::disable
```

Example Use the following example to disable HTTP processing.

```
when CLIENT_ACCEPTED {
    TCP::collect 7
}
when CLIENT_DATA {
    if { [TCP::payload 7] equals "CONNECT" } {
        SSL::disable
    }
    TCP::release
}
when HTTP_REQUEST {
    if { [HTTP::method] equals "CONNECT" } {
        log "A HTTP CONNECT was received."
        HTTP::respond 200 content OK
        HTTP::disable
        SSL::enable
        SSL::collect
    }
}
```

Example Use the following example to disable HTTP request processing.

```
when HTTP_REQUEST {
    HTTP::disable
    log "Work with SSL Proxy or TCP (generic) "
    node 192.168.80.81 80
}
```

Example Use the following example to disable HTTP response processing.

```
when HTTP_RESPONSE {
    HTTP::disable
    log "Ignore HTTP processes after this point."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- SERVER_CONNECTED
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERHELLO

HTTP::fallback

Description This command will specify or override the fallback host that is specified in the HTTP profile.

Syntax

```
HTTP::fallback <host>
```

Example Use the following example to specify the fallback host in HTTP profile.

```
when LB_FAILED {
    HTTP::fallback "http://backup.example.com"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA

HTTP::header

Description This command will query for or manipulate an HTTP header.

Syntax

```
HTTP::header [value] <name>
```

This will return the value of the HTTP header named `<name>`. Drop the `<value>` argument when the header name does not enter any conflicts with subcommands.

```
HTTP::header names
```

This will return a list of all the headers present on the request or response.

```
HTTP::header count
```

This will return the number of HTTP headers present in the request or response.

```
HTTP::header at <index>
```

This will return the HTTP header that the ACOS device finds at the zero-based index value.

```
HTTP::header exists <name>
```

This will return true if the named header is present on the request or response.

```
HTTP::header insert ["lws"] <name> <value>
```

This will insert the named HTTP header and its value into the end of the HTTP request or response. If "lws" is specified, the ACOS device adds linear white space to long header values.

```
HTTP::header insert ["lws"] {n1, v1, n2, v2, n3, v3, …}
```

This will pass a Tcl list to insert into a header. In this situation, the ACOS device will treat the list as a list of name/value pairs. If "lws" is specified, the ACOS device adds linear white space to long header values.

```
HTTP::header [value] <name> <string>
```

This will set the value of the named header. When there is a present header, the command will replace the header; In other situations, the command will add the header. Drop the <value> argument if the header name does not collide with any other values.

```
HTTP::header replace <name> [<string>]
```

This will replace the last occurrence of the named header with the string <string>. It performs a header insertion when the header was not present.

```
HTTP::header remove <name>
```

This will remove all headers names with named <name>.

```
HTTP::header sanitize <header name>+
```

This will remove everything except the headers specified. It does not remove essential HTTP headers, though.

```
HTTP::header at <index> [nvp]
```

This will return the HTTP header that the ACOS device finds in at the zero-based index value. The nvp option will return the entire header as a name-value-pair (NVP).

```
HTTP::header values <name>
```

This will return the value or values of the HTTP header named <name>.

NOTE:

- If both HTTP::cookie and HTTP::header commands are used to modify the same header, then HTTP::cookie takes precedence. When there is a single value for the HTTP header, that value is returned. When there are multiple headers with the same name, the command returns the last value from all of them. If it is required to check all HTTP headers that include multiple headers of the same name, use HTTP::header at <index> nvp.
- To check if an HTTP header exists or not, use the `HTTP::header exists <name>` command. For example, `HTTP::header exists "foo"` will return true if an HTTP header exists and false if it doesn't exist. If "exists" in the above command is misspelled as "exist", then ACOS will interpret this command differently and insert a header called "exist" with the string "foo" as `HTTP::header [value] <name> <string>`.

Example Use the following example to remove all headers names with the specified name.

```
when HTTP_REQUEST {
    if { [HTTP::header exists "Accept-Encoding"] } {
        HTTP::header remove "Accept-Encoding"
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::host

Description This command will return the host name of the HTTP request.

Syntax

```
HTTP::host
```

Example Use the following example to return the host name of the HTTP request.

```
when HTTP_REQUEST {
    if { [HTTP::host] starts_with "secure"} {
        HTTP::redirect "https://[HTTP::host][HTTP::uri]"
    }
}
```

```
when ICAP_RESPONSE {
if { [HTTP::host] starts_with "secure"} {
HTTP::redirect "https://[HTTP::host][HTTP::uri]"
 }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::is_keepalive

Description This command will return a true value when it is a Keep-Alive connection.

Syntax

```
HTTP::is_keepalive
```

Example Use the following example to return a value for a Keep-Alive connection.

```
when HTTP_RESPONSE {
    if { not ([HTTP::is_keepalive]) } {
        HTTP::close
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::is_redirect

Description This command will return a true value if the response is a redirect of a certain type.

Syntax

```
HTTP::is_redirect
```

Example Use the following examples to log the message with a value for a certain type of redirect.

```
when HTTP_RESPONSE {
    if { [HTTP::is_redirect] } {
        log "This is the server redirect value:"
    }
}
```

```
when ICAP_RESPONSE {
if { [HTTP::is_redirect] } {
log "This is the server redirect value:"
}
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::method

Description This command will return the type of HTTP request method.

Syntax

```
HTTP::method
```

Example Use the following example to log the message with a value for certain type of redirect.

```
Example 1:
when HTTP_REQUEST {
    log "This is the HTTP method: [HTTP::method]"
}
Example 2:
when ICAP_RESPONSE {
 {
log "This is the HTTP method: [HTTP::method]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::password

Description Returns the password from HTTP basic authentication.

Syntax

```
HTTP::password
```

Example Use the following example to return the password from HTTP basic authentication.

```
[AFLEX_NSCMDID_HTTP_PASSWORD] = {
"HTTP::password",
A10Tcl_HTTP_PasswordObjCmd,      A10TclCompileHTTP_
PasswordCmd,
```

```
AFLEX_VPORT_BITS_HTTP,
{0},
0
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA

HTTP::path

Description This command will return the path part of the HTTP request.

Syntax

```
HTTP::path [<string>]
```

Example Use the following examples to return the path part of the HTTP request.

```
when HTTP_REQUEST {
    log "This is the host HTTP: [HTTP::host]"
    log "This is the path of HTTP: [HTTP::path]"
}
```

```
when HTTP_REQUEST {
    if { [HTTP::path] equals "/" } {
        HTTP::redirect "https://[HTTP::host]/exchange/"
    } else {
        pool example_service-group
    }
}
```

```
when ICAP_RESPONSE {
log "This is the host HTTP: [HTTP::host]"
log "This is the path of HTTP: [HTTP::path]"
```

```
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::payload

Description This command will query for or replace content information. It allows retrieval of content, queries for content size, or replacement for a certain amount of content.

Syntax

```
HTTP::payload [<size>]
```

This will return the content that the `HTTP::collect` command has collected by time of the request. If no size is specified, the system will return the collected content.

```
HTTP::payload length
```

This will return the size of the content that the command has collected by time of the request, but without the HTTP headers.

```
HTTP::payload <offset> <size>
```

This will return the content that the `HTTP::collect` command has collected, starting at `<offset>` with size equals `<size>`.

```
HTTP::payload replace <offset> <size> <string>
```

This will replace the amount of content that is specified using the `<size>` argument, starting at `<offset>` with `<string>`.

Example Use the following example to the content that the HTTP::collect command has collected by time of the request.

```
when HTTP_RESPONSE {
    HTTP::collect [HTTP::header Content-Length]
}
```

```
when HTTP_RESPONSE_DATA {
    regsub "Internal Site" [HTTP::payload] "Public Site"
newpayload
    log "We have changed the payload to reflect a Public Site"
    HTTP::payload replace 0 [HTTP::header Content-Length]
$newpayload
    HTTP::release
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::query

Description This command will return the query part of the HTTP request.

Syntax

```
HTTP::query
```

Example Use the following example to log the message to the query of the HTTP request.

```
when HTTP_REQUEST {
    log "This is the HTTP path: [HTTP::path]"
    log "This is our HTTP query: [HTTP::query]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::redirect

Description This command will redirect an HTTP request or response to the specified URL.

Syntax

```
HTTP::redirect <url>
```

NOTE: This command will send the response to the client immediately. It cannot be specified multiple times in an aFleX script, nor can commands that modify header or content be specified after this command, due to its functionality.

Example Use the following example to redirect an HTTP response to the specified URL.

```
when HTTP_RESPONSE {
    if { [HTTP::status] == 404 } {
        HTTP::redirect "http://backup.example.com"
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::release

Description This command will release the collected data. Unless a subsequent `HTTP::collect` command was issued, the `HTTP::release` command inside of the HTTP_REQUEST_DATA and HTTP_RESPONSE_DATA events is unnecessary, since in these situations, the data is implicitly released.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax `HTTP::release`

Example Use the following example to release the collected data.

```
when HTTP_RESPONSE {
    HTTP::collect [HTTP::header Content-Length]
}
when HTTP_RESPONSE_DATA {
    regsub "Internal Site" [HTTP::payload] "Public Site"
newpayload
    log "We have changed payload to reflect Public Site"
    HTTP::payload replace 0 [HTTP::header Content-Length]
$newpayload
    HTTP::release
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::request

Description This command will return the raw request header string. Access the request payload using the `HTTP::collect` command.

Syntax `HTTP::request`

Example Using this example will return the raw request header string. It uses the `HTTP::method` and the HTTP version. It demonstrates the generation of identical results for both log entries.

```
when HTTP_REQUEST {
    log "This is the HTTP request: [HTTP::method] [HTTP::uri]
HTTP/[HTTP::version]"
    log "This is the HTTP request: [HTTP::request]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::request_num

Description This command will return the number of HTTP requests that a client made on the connection.

Syntax

```
HTTP::request_num
```

Example Use the following example to returns the number of HTTP requests that a client made on the connection.

```
when HTTP_REQUEST {
    log "This is the Request #: [HTTP::request_num]"
}
```

```
when ICAP_RESPONSE {
{
log "This is the Request #: [HTTP::request_num]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::respond

Description This command will allow users to generate or rewrite a client request or a server response. It is a powerful API that gives users the ability to generate or rewrite a client request or a server response

Upon execution of the command on the client side, it will send the response to the client without any load balancing taking place.

Upon execution of the command on the server side, the content from the actual server will be discarded and replaced with information provided to this API.

Syntax

```
HTTP::respond <status code> [content <content Value>] [<Header name> <Header Value>]+
```

NOTE: The maximum size response for this command that can be sent is 64 KB.

NOTE: No further aFlex scripts should be run after this API due to the functionality of this command.

Example Use the following example to generate the client request and a server response.

Example Use of the following example sends a redirect with a cookie set.

```
when HTTP_REQUEST {
    set cookie [format "%s=%s; path=/; domain=%s" CookieName CookieValue ".example.com"]
    HTTP::respond 302 Location "https://www.example.com" "Set-Cookie" $cookie
}
```

Or it can be used so it sends an apology page from with in the aFleX.

```
when HTTP_REQUEST {
        HTTP::respond 200 content "<html><head><title>Apology Page</title></head><body>We are sorry for the inconvenience, but the site is temporarily out of service<br>If you feel you have reached this page in error, please try again.<p></body></html>"
}
```

Use the following example to generate the ICAP response.

```
when ICAP_RESPONSE {
{
```

```
HTTP::respond 200 content "<html><head><title>Apology Page</
title></head><body>We are sorry for the inconvenience, but the
site
is temporarily out of service<br>If you feel you have reached
this
page in error, please try again.<p></body></html>"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::retry

Description This command will send an HTTP request to the server. It also triggers the `HTTP_REQUEST` event.

Syntax

```
HTTP:retry
```

NOTE: The HTTP retry command is supported only for virtual port types HTTP and HTTPS. Fast-HTTP or other virtual port types are not supported.

Example Use the following example to send an HTTP request to the server.

```
when HTTP_RESPONSE {
    if { [HTTP::status] == 503 } {
        HTTP::retry
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::scheme

Description This command retrieve the scheme (protocol) part of an HTTP request, which indicates whether the request is using HTTP or HTTPS.

Syntax

```
HTTP::scheme
```

Example Use the following example to check the scheme of the incoming HTTP request and log a message indicating whether the request is secure or not.

```
when HTTP_REQUEST {
    set scheme [HTTP::scheme]
    if { $scheme eq "http" } {
        log "Insecure HTTP request received"
    }
}
```

HTTP::status

Description This command will return the response status code.

Syntax

```
HTTP::status
```

Example Use the following example to return the HTTP response status code.

```
when HTTP_RESPONSE {
    if { [HTTP::status] == 404 } {
        HTTP::redirect "http://backup.example.com"
    }
}
```

Example Use the following example to return the ICAP response status code.

```
when ICAP_RESPONSE {
if { [HTTP::status] == 404 } {
HTTP::redirect "http://backup.example.com"
}
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- ICAP_RESPONSE

HTTP::stream

Description This command will replace the specified string of an HTTP response.

The content returned by the server can be of either a content-length or chunked header format. Whatever the response from the server, content-length or chunk-encoded header, the response format after a string replacement will always be chunk-encoded (a Transfer-Encoding: chunked header).

Syntax

```
HTTP::stream replace <old_string> <new_string>
```

NOTE: The `HTTP::stream` command can execute up to 32 instances of multiple string replacements in this current release.

Example Use the following example to replace the specified string of an HTTP response.

```
when HTTP_RESPONSE {
    HTTP::stream replace "Internal Site" "Public Site"
    HTTP::stream replace "http://" "https://"
}
```

Example Use the following example to replace the specified string of an ICAP response.

```
when ICAP_RESPONSE {
HTTP::stream replace "Internal Site" "Public Site"
HTTP::stream replace "http://" "https://"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

HTTP::uri

Description This command will return or set the URI of the request. This command replaces the `http_uri` command.

Syntax

```
HTTP::uri <string>
```

This will change the URI passed to the server. Check that it starts with a slash. The URI string does not include the http or https protocol or hostname.

Example Use the following example to change the URI passed to the server.

```
when HTTP_REQUEST {
    if { [HTTP::uri] ends_with ".html" } {
        pool service_group_static
    } elseif { [HTTP::uri] ends_with ".asp" } {
        pool service_group_dynamic
    }
}
```

Example Use the following example to change the URI passed to the ICAP response.

```
when ICAP_RESPONSE {
if { [HTTP::uri] ends_with ".html" } {
pool service_group_static
} elseif { [HTTP::uri] ends_with ".asp" } {
pool service_group_dynamic
        }
}
```

Valid Events

- HTTP_REQUEST

- HTTP_REQUEST_DATA
- ICAP_RESPONSE

HTTP::username

Description This command will returns the username from HTTP basic authentication..

Syntax

```
HTTP:username
```

Example Use the following example to return the username from HTTP basic authentication.

```
when HTTP_REQUEST {
    if {[HTTP::username] eq "admin"} {
        log "Admin user accessed"
    }
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT
- HTTP_REQUEST
- HTTP_REQUEST_DATA

HTTP::version

Description This command will return or set the HTTP version of the request or response. It replaces the `http_version` command.

Syntax

```
HTTP::version ["0.9" | "1.0" | "1.1"]
```

Example Use the following example to sets the HTTP version of the response.

```
when HTTP_RESPONSE {
    log "This is the version of HTTP: [HTTP::version]"
```

```
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

ICAP Commands

The following IP commands are supported:

- ICAP::disable
- ICAP::header add
- ICAP::header remove
- ICAP::header values
- ICAP::header replace
- ICAP::header replace-all
- ICAP::method
- ICAP::status
- ICAP::respmod_valid
- ICAP::reqmod_valid
- ICAP::uri

For information about aFleX commands, see aFleX Commands.

For information about ICAP events, see ICAP Events.

ICAP::disable

Description This command will disable ICAP for certain requests, based on the HTTP headers.

Syntax

```
ICAP::disable
```

Example Use of the following example selects a specific pool for a specific client IP address.

```
when HTTP_REQUEST {
    set method [HTTP::method]
    if { ($method matches "POST")
    or ($method matches "PUT") } {
    return // follow the ICAP policy configured with CLI
    } else {
    ICAP::disable // disable ICAP template policy
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_RESPONSE

ICAP::header add

Description This command inserts a header to ICAP reqmod/respmod packet

Syntax

```
ICAP::header add <attr_name> <attr_value>
```

NOTE: For singleton attributes only the first one added will appear in traffic. For others, values will be separated by comma.

Example Add multiple times to header allowing list (Upgrade).

```
when ICAP_REQUEST {
        ICAP::header add Upgrade h2
        ICAP::header add Upgrade h2c
        ICAP::header add "X-DEF" abc
```

```
        ICAP::header add "X-DEF" fhga
        ICAP::header add "X-CLIENT-IP" 135
        ICAP::header add Preview 2
        ICAP::header add Preview 3
        }
```

Valid Events

- ICAP_REQUEST

ICAP::header remove

Description This command will remove default, non-default, and previously header values.

Syntax

```
ICAP::header remove <attr_name>
```

Example Use the following example to remove specific header values.

```
when ICAP_REQUEST {
 ICAP::header remove X-Unknown
 ICAP::header remove X-CLIENT-IP
 ICAP::header add X-DEF aaaaaa
 ICAP::header remove X-DEF
    }
```

Valid Events

- ICAP_REQUEST

ICAP::header replace

Description The replace and add command are similar except that replace will not append value to list, it will replace the existing values.

Syntax

```
ICAP::header replace <attr_name> <attr_value>
```

Example Use the following example to replace the specified header with given value.

```
when ICAP_REQUEST {
 ICAP::header replace Preview 2
```

```
 ICAP::header replace Preview 3
 ICAP::header replace X-CLIENT-IP replaced
 ICAP::header add X-DEF abc
 ICAP::header replace X-DEF def
}
```

Valid Events

- ICAP_REQUEST

ICAP::header replace-all

Description This command will replace existing header values.

Syntax

```
ICAP::header replace-all <header_text>
```

Example Use the following example to replace the whole header with given text.

```
when ICAP_REQUEST {
   ICAP::header replace-all "Host: 20.20.5.10:1344\r\nDate:
Tue, 28-May-2019 09:17:50 GMT\r\nEncapsulated:
req-hdr=0, req-body=147\r\nPreview: 1\r\nAllow:
204\r\nX-Client-IP: 20.20.3.10\r\nX-Server-IP: 20.20.5.10\r\n"
ICAP::header add X-DEF abc
}
```

Valid Events

- ICAP_REQUEST

ICAP::header values

Description This command can get a header value from ICAP reqmod/respmod response.

Syntax

```
ICAP::header values <attr_name>
```

Example Use the following example to get a header value from ICAP respmod.

```
when ICAP_RESPONSE {
log " ISTag header value is [ICAP::header values ISTag]"
}
```

Valid Events

- ICAP_RESPONSE

ICAP::method

Description This command returns ICAP request method which can be reqmod or respmod.

Syntax

```
ICAP::method
```

Example Use the following example to return the method of this request.

```
when ICAP_REQUEST {
        log "method [ICAP::method]"
        log "get uri [ICAP::uri]"
                                  }
```

Valid Events

- ICAP_REQUEST

ICAP::reqmod_valid

Description This command will check if reqmod-icap template is bound under the vPort and the ICAP service used is active. Return 1 only when reqmod-icap template is bound and the ICAP service used is active; otherwise, return 0.

Syntax

```
ICAP::reqmod_valid
```

Example Use the following example to check that vport has reqmod configured and that it is not disabled.

```
when HTTP_REQUEST {
   log "req [ICAP::reqmod_valid]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA

ICAP::respmod_valid

Description This command will check if respmod-icap template is bound under the vPort and the ICAP service used is active. Return 1 only when respmod-icap template is bound and the ICAP service used is active; otherwise, return 0.

Syntax

```
ICAP::respmod_valid
```

Example Use the following example to check that vport has respmod configured and that it is not disabled.

```
when HTTP_RESPONSE {
   log "resp [ICAP::respmod_valid]"
}
```

Valid Events

- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

ICAP::status

Description This command will get ICAP response status code.

Syntax

```
ICAP::status
```

Example Use the following example to return the status code of the response.

```
when ICAP_RESPONSE {
log "status [ICAP::status]"
}
```

Valid Events

- ICAP_RESPONSE

ICAP::uri

Description This command will set or return ICAP service URI sent to ICAP server.

Syntax

```
ICAP::uri
```

Syntax

```
ICAP::uri <uri>
```

Example Use the following example to get the uri of the request.

```
when ICAP_REQUEST {
ICAP::uri icap://A10icap:1344/echo
}
```

Valid Events

- ICAP_REQUEST

IP Commands

The following topics are covered in this section:

- IP::addr
- IP::category
- IP::client_addr
- IP::local_addr
- IP::protocol
- IP::remote_addr
- IP::reputation
- IP::server_addr
- IP::stats
- IP::tos
- IP::ttl
- IP::version

For information about aFleX commands, see aFleX Commands.

For information about IP events, see IP, TCP, and UDP Events.

IP::addr

Description This command will compare IP address/subnet/supernet to IP address/subnet/supernet. It returns 0 if there is no match, and 1 in case there is a match.

Syntax

```
IP::addr <addr1>[/<mask>] equals <addr2>[/<mask>]
```

```
IP::addr
```

NOTE: The IP::addr command does not perform a string comparison. If a literal string comparison is needed, compare the 2 strings with the appropriate operator (for example, equals, contains, starts_with) instead of using this command.

Example Using this example, a comparison will be done with IP address 192.168.10.1 with subnet 192.168.0.0/16. This will return a value of 1 since there is a match.

```
[IP::addr 192.1680.10.1 equals 192.168.0.0/16]
```

Example Use this example to compare a client-side IP address with subnet 192.168.0.0/16. The client IP address will determine whether a 1 or 0 value is returned.

```
[IP::addr [IP::client_addr] equals 192.168.0.0/16]
```

Example Use of the following example selects a specific pool for a specific client IP address.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10]  } {
        pool example_service_group
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST

- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::category

Description This command fetches the IP category from a local database.
A TCL contains the following category list:

- spam-sources
- windows-exploits
- web-attacks
- botnets
- scanners
- dos-attacks
- reputation
- phishing
- proxy
- mobile-threats
- tor-proxy
- uncategorized

Syntax

```
IP::category IP
```

In this command, IP is the IP address for which the categories need to be fetched from the database.

Example Use the following example to return the IP category string from threat-intel local database.

```
when HTTP_REQUEST {
      set local_ip [IP::local_addr]
      set cat_list [IP::category $local_ip]
      foreach cat $cat_list {
          log "IP category: $cat"
      }
}
```

Valid Events

- CLIENT_ACCEPTED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- DNS_REQUEST
- DNS_RESPONSE

IP::client_addr

Description This command will return the client IP address of a connection. It is the same as using the command `clientside { IP::remote_addr }`.

Syntax

```
IP::client_addr
```

Example Use the following example to select a specific service group for a specific client IP address.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.1.10]  } {
        pool example_service_group
    }
}
```

Valid Events

- CLIENT_ACCEPTED

- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::local_addr

Description This command is useful for addressing generic rules that are reused. It is also useful in reusing the connected endpoint in another statement or in making routing type decisions. The `IP::client_addr` and `IP::server_addr` commands can also be specified.

Syntax

```
IP::local_addr
```

This will return the IP address of the ACOS being used in the connection. From the clientside position, this is the destination IP address (virtual IP address). From the serverside position, this is the source IP address. The following example shows the SNAT address if SNAT is used, otherwise it spoofs client IP address).

Example Use the following example to select a specific service group for a specific virtual IP address.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::local_addr] equals 192.168.1.10] } {
        pool service_group_internal
    } else {
        pool example_service_group
    }
}
```

Example Use the following example to select a specific SNAT IP address.

```
when SERVER_CONNECTED {
    log "This is the SNAT IP address [IP::local_addr] that has
been assigned to [IP::client_addr]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::payload

Description This command is used to directly access or modify the IP layer payload.

Syntax

```
IP::payload
```

The IP address for which the payload needs to be fetched from the database.

Example Use the following example to log the raw IP payload data received from the client:

```
when CLIENT_DATA {
    set payload [IP::payload]
    log "Raw IP Payload: $payload"
}
```

IP::protocol

Description This command will return the IP protocol value.

Syntax

```
IP::protocol
```

Example Use the following example to select a specific service group based on IP protocol version.

```
when CLIENT_ACCEPTED {
    if { [IP::protocol] == 6 } {
        pool service_group_tcp
    } else {
        pool service_group_udp
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::remote_addr

Description This command will return the IP address of the host at the far end of the connection. From the clientside position, this is the client IP address. From the serverside position, this is the node IP address. The `IP::client_addr` and `IP::server_addr` commands can also be specified.

Syntax

```
IP::remote_addr
```

Example Use the following example to select a specific service group for a specific client IP address. Then log the server address of the real server where the request is to be forwarded.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::remote_addr] equals 192.168.1.10]  } {
        pool example_service_group
    }
}
when SERVER_CONNECTED {
    log "This is the node IP address [IP::remote_addr]
assigned to [IP::client_addr]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::reputation

Description This command fetches the IP reputation value from a local database. It can display one of the following values:

100	Best Reputation
1-20	High Risk
21-40	Suspicious
41-60	Moderate Risk
61-80	Low Risk
81-100	Trustworthy
0	No Data Available

Syntax

```
IP::reputation IP
```

In this command, IP is the IP address for which the reputation value needs to be retrieved.

NOTE: There are only 'high risk' records in the local database.

Example Use the following example to returns the IP reputation value from local database.

```
when HTTP_REQUEST {
set local_ip [IP::local_addr]
log "Access Server $local_ip (reputation: [IP::reputation
$local_ip])"
}
```

Valid Events

- CLIENT_ACCEPTED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- DNS_REQUEST
- DNS_RESPONSE

IP::server_addr

Description This command will return the server's (node's) IP address, after a server side connection has been established. It is the same as using the server side command {IP::remote_addr}. The command will return a value of 0 if no server side connection has been made.

Syntax

```
IP::server_addr
```

Example Use the following example to log the end node or the real server address.

```
when SERVER_CONNECTED {
    log "This is the node IP address [IP::server_addr]"
}
```

Valid Events

- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

IP::stats

Description This command will supply information regarding the number of packets or bytes being sent or received in a given connection.

Syntax

```
IP::stats pkts [ in | out ]
```

This will return the number of packets in or out. If neither is specified, it will return the total number of packets in and packets out.

```
IP::stats bytes [ in | out ]
```

This will return number of bytes in or out. If neither is specified, it will return total number of bytes in and bytes out.

Example Use the following example to log the total received packets for the connection.

```
when CLIENT_CLOSED {
        log "Total received packets: [IP::stats pkts in]"
}
```

Valid Events

- All.
- For information about aFleX events, see aFleX Events.

IP::tos

Description This command will select a different pool of servers based on the Type of Service (ToS) level within a packet. The ToS standard is one method where network equipment can identify and treat traffic differently based on an identifier. As soon as traffic enters the site, the ACOS device can apply a rule that sends traffic to different pools of servers based on the ToS level within a packet.

Syntax

```
IP::tos
```

This will select a different pool of servers based on the ToS level within a packet.

Example Use the following example to select a specific pool based on TOS level in the packet.

```
when CLIENT_ACCEPTED {
    if { [IP::tos] == 16 } {
        pool service_group_priority
    } else {
        pool example_service_group
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::ttl

Description This command will return the TTL of the current packet being acted upon.

Syntax

```
IP::ttl
```

Example Use the following example to drop the connection if the TTL for the packet is below 3.

```
when CLIENT_ACCEPTED {
        if { [IP::ttl] < 3 } {
                drop
        }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA

- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

IP::version

Description This command will return the version of the current packet being acted upon.

Syntax

```
IP::version
```

Example Use the following example to select a specific service group based on IP protocol version.

```
when CLIENT_ACCEPTED {
    if { [IP::version] == 6 } {
        pool service_group_ipv6
    } else {
        pool service_group_ipv4
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_SELECTED

- SERVER_CLOSED
- SERVER_CONNECTED
- CLIENT_DATA
- SERVER_DATA

Limit ID Commands

The following Limit ID (LID) commands are supported:

- LID::conn_limit
- LID::conn_rate_limit
- LID::exists
- LID::nat_pool
- LID::request_limit
- LID::request_rate_limit
- LID::type

For information about aFleX commands, see aFleX Commands.

NOTE: Multiple LID definitions may be available for a non-global LID. This includes a LID in a policy template bound to a virtual port, a LID in DNS template bound to a virtual port, a LID in a policy template bound to a virtual server, and a LID configured in a system-wide policy template.

NOTE: To apply these commands, the LID must be configured and attached to the same virtual port as the aFleX policy using the template. If GLID is used, it must be configured and enabled on the configuration.

LID::conn_limit

Description Returns a list of conn-limit and LID type, each one for a matching LID where conn-limit is configured.

Syntax

```
LID::conn_limit <lid-id>
```

Example Use the following example to log the connection limit specified for LID 1.

```
when HTTP_REQUEST {
    log "The LID connection limit for LID1 is [LID::conn_limit lid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED

- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::conn_rate_limit

Description Returns a list of conn-rate-limit values and LID type, one each for a matching LID where conn-rate-limit is configured.

Syntax

```
LID::conn_rate_limit <param>
```

Example Use the following example to log the connection rate limit specified for GLID 1.

```
when HTTP_REQUEST {
    log "The LID connection rate limit for glid1 is
[LID::conn_rate_limit glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE

- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::exists

Description Returns a Boolean value that indicates whether the specified LID exists.

Syntax

```
LID::exists <lid-id>
```

Example Use the following example to log the presence of the specified GLID.

```
when HTTP_REQUEST {
    log "The LID exists for glid1 [LID::exists glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::nat_pool

Description Returns a list of string and LID type, one each for a matching LID where nat-pool is configured.

Syntax

```
LID::nat_pool <lid-id>
```

Example Use the following example to log the NAT pool associated with GLID 1.

```
when HTTP_REQUEST {
    log "The LID NAT pool for glid1 is [LID::nat_pool glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST

- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::request_limit

Description Returns a list of request-limit and LID type, one each for a matching LID where request-limit is configured.

Syntax

```
LID::request_limit <param>
```

Example Use the following example to log the request limit specified for GLID 1.

```
when HTTP_REQUEST {
    log "The LID request limit for glid1 is [LID::request_
limit glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::request_rate_limit

Description Returns a list of request-rate-limit values and LID type, one each for a matching LID where conn-rate-limit is configured.

Syntax

```
LID::request_rate_limit <param>
```

Example Use the following example to log the request rate limit specified for GLID 1.

```
when HTTP_REQUEST {
    log "The LID request rate limit for glid is [LID::request_
rate_limit glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

LID::type

Description Returns a list of LIDs of the specified type.

Syntax

```
LID::type <param>
```

Example

Use this example to return a list of LID types, one each for a matching LID. The type can be one of the following: `global`, `vport-policy`, `vport-dns`, `vserver-policy`, `system-policy`.

```
when HTTP_REQUEST {
        log "The glid1 type is [LID::type glid1]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- DNS_REQUEST
- DNS_RESPONSE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Link Commands

The following link commands are supported:

- LINK::lasthop
- LINK::nexthop
- LINK::vlan_id

For information about aFleX commands, see aFleX Commands.

LINK::lasthop

Description Returns the MAC address of the last hop.

Syntax

```
LINK::lasthop
```

Example Use the following example to return the MAC address of the last hop.

```
when HTTP_REQUEST {
    log "The LID request rate limit for glid1 is
[LID::request_rate_limit glid1]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

LINK::nexthop

Description Returns the MAC address of the next hop.

Syntax

```
LINK::nexthop
```

Example Use the following example to return the MAC address of the next hop.

```
when SERVER_CONNECTED {
    log "The Ethernet is { [LINK::lasthop] to [LINK::nexthop]
tag is [LINK::vlan_id] }"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

LINK::vlan_id

Description Returns the VLAN tag of the packet. In some cases, the VLAN ID may be unavailable. In these cases a value of 0 will be returned.

Syntax

```
LINK::vlan_id
```

Example Use the following example to return the VLAN tag of the packet.

```
when CLIENT_ACCEPTED {
    set log_message "Client is { [IP::client_addr]:
[TCP::client_port] -> [IP::local_addr]:[TCP::local_port] }"
    append log_message " Ethernet is { [string range
[LINK::lasthop] 0 16] -> [string range [LINK::nexthop] 0 16]"
    append log_message " Tag is [LINK::vlan_id] }"
    log "$log_message"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

Load-balancing Commands

The following load-balancing (LB) commands are supported:

- LB::down
- LB::reselect
- LB::server
- LB::status

For information about aFleX commands, see aFleX Commands.

For information about events related to load-balancing, see Global Events.

LB::down

Description Temporarily marks the current real port down for 30 seconds.

Syntax

```
LB::down
```

Example See Example 2 in LB::reselect.

Valid Events

- LB_FAILED
- LB_SELECTED

LB::reselect

Description Reperforms server selection.

Syntax

```
LB::reselect [pool <pool-name> [<member>]]
```

Causes SLB to select the next available member (server and port) from the same service group used for the initial server selection. To specify the service group to use, use the `pool <pool-name>` option. If you also use the `<member>` option, the specified member is selected from the specified service group.

NOTE: This command applies to Layer 7 traffic only for HTTP and HTTPS.

NOTE: Failure to execute this command will not always trigger the `LB_FAILED` event.

NOTE: Server template limits are applied for both service-group and server selection. Commands that call for server selection (i.e., `node`, `pool`, `persist`, etc.) will enforce server template limits on the selected server. As a result, new connections that match a persist uie entry may be unable to use the rport and a default server selection will occur instead. To prevent default server selection, use the no def-selection-if-pref-failed command for the virtual port.

Example In this aFleX policy, the `HTTP::retry` command retries sending a client's request to a service port that replies with an HTTP 5xx status code. If the first server continues to reply with a 5xx status code after 3 retries, the `LB::reselect` command reassigns the client request to another server.

```
when CLIENT_ACCEPTED {
    set retry 0
    set max_retry 3
    set reselect 0
}
when LB_SELECTED {
    if { $retry > 0 } {
        LB::reselect
        incr reselect
    }
}
when HTTP_RESPONSE {
    if { $retry < $max_retry } {
        if { [HTTP::status] starts_with "5" } {
            incr retry
        }
    }
}
```

Example This aFleX policy is similar to the one above, except the `LB::down` command in the policy marks the service port down for 30 seconds.

```
when CLIENT_ACCEPTED {
    set retry 0
    set max_retry 3
}
when HTTP_REQUEST {
    log "HTTP_REQUEST: Retry Count: $retry"
}
when LB_SELECTED {
    log "LB_SELECTED: Current Retry Count: $retry"
    if { $retry > 0 } {
        log "LB::reselect"
        LB::down
```

```
        LB::reselect
    }
}
when HTTP_RESPONSE {
    log "HTTP_RESPONSE: [HTTP::status]"
    if { $retry < $max_retry } {
        if { [HTTP::status] starts_with "5" } {
            log "HTTP::retry"
            incr retry
            HTTP::retry
        }
    }
}
```

Valid Events

- LB_FAILED
- LB_SELECTED

LB::server

Description Returns the results of pool and node selection.

Syntax

```
LB::server
```

Returns a Tcl list containing the pool, node, node IP address, and Layer 4 protocol port selected by SLB. If no server was selected when the script was executed, or all servers are down, the command returns only the default pool name.

```
LB::server pool
```

Returns the pool of the currently selected member. If no server was selected when the script was executed, or all servers are down, the command returns only the default pool name.

```
LB::server addr
```

Returns the IP address of the currently selected pool member. If no server was selected when the script was executed, or all servers are down, the command returns null.

```
LB::server port
```

Returns the port of the currently selected pool member. If no server was selected when the script was executed, or all servers are down, the command returns null.

```
LB::server name
```

Returns the name of the currently selected pool member. If no server was selected when the script was executed, or all servers are down, the command returns null.

```
LB::server resolve <server-name>
```

Returns the IPv4/IPv6 address of the specified server. If no such server exists, an empty string is returned.

```
LB::server resolve addr { <ipv4-address> | <ipv6-address> }
```

Returns the name of the server with the specified IPv4 or IPv6 address. Returns an empty string if no server with the specified IP address exists.

Example The following example shows a script that replaces the Host header with a header that contains the backend server's hostname:

```
when LB_SELECTED {
    switch [LB::server addr] {
        "192.168.2.16" { HTTP::header replace Host
server1.example.com }
        "192.168.2.18" { HTTP::header replace Host
server2.example.com }
    }
}
```

Example This examples shows a script which checks if the default pool has less than 2 active members.

```
when HTTP_REQUEST {
    if { [active_members [LB::server pool]] < 2 } {
        HTTP::respond 200 content "We are sorry, but the site
you are looking for is temporarily out of service."
    }
}
```

Example The following example show a script that logs server names with their associated IP addresses.

```
when CLIENT_DATA {
    log "The LB Server resolve of 192.168.80.82 is [LB::server
resolve addr 192.168.80.82]"
    log "The LB server resolve of rs1 is [LB::server resolve
name rs1]"
    log "[LB::server resolve addr 2001:DB8::a10]"
    log "[LB::server resolve name rs1]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

NOTE: The `LB::server resolve [addr]` option is valid with all events.

LB::status

Description Returns the health check status (up or down) of a node or pool.

Syntax

```
LB::status node <ipaddr> [port <port-num> {tcp | udp}]
```

If you were to specify the node IP address only, the Layer 3 health status of the server is returned. If you also specify a protocol port and

its transport protocol, the health status of the port is also returned. If you use the `port` option, the port number and the transport protocol are required.

```
LB::status pool <pool_name>
```

Returns the health status of the service group.

```
LB::status pool <pool_name> member <ipaddr>
```

Returns the health status of the specified member (node).

```
LB::status pool <pool_name> member <ipaddr> <port_num>
```

Returns the health status of the specified service port.

Example Use the following example to check the health check status of a node.

```
when HTTP_REQUEST {
    if { [LB::status node 192.168.80.82 port 80 tcp] equals
"up" } {
        log "node 192.168.80.82 port 80 is UP!"
    } else {
        log "node 192.168.80.82 port 80 is DOWN!"
    }
}
```

Example Use the following example to check the health status of the service group.

```
when HTTP_REQUEST {
    if { [LB::status pool example_service_group 192.168.80.82
80] equals "up" } {
        log "The member 192.168.80.82 port 80 of service group
example_service_group is UP!"
    } else {
        log "The member 192.168.80.82 port 80 of service group
example_service_group is DOWN!"
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

MQTT Commands

The following MQTT commands supported are:

- MQTT::clean_session_flag
- MQTT::client_id
- MQTT::collect
- MQTT::drop
- MQTT::dup_flag
- MQTT::keep_alive
- MQTT::length
- MQTT::packet_id
- MQTT::password
- MQTT::payload
- MQTT::payload_length
- MQTT::protocol_name
- MQTT::protocol_version
- MQTT::qos
- MQTT::replace
- MQTT::respond
- MQTT::retain_flag
- MQTT::return_code
- MQTT::return_code_list
- MQTT::session_present_flag
- MQTT::topic
- MQTT::type

- MQTT::username
- MQTT::will

NOTE:

- Default and Client_id based load balancing methods are supported on MQTT vPort.Clientid-hash-persist first N: Use the first N bytes for server selection.Clientid-hash-persist last N: Use the last N bytes for server selection.
- Clientid-hash-persist offset N: Start from Nth bytes of the client id.
- Must be used together with first or last option.
- aFleX processes a message within 1MB only.
- For bigger messages, aFleX forwards the message successfully, but only prints message contents and flags up to 1MB. For example:
  - If an MQTT CONNECT message is with a large will-topic (example:2000 bytes) following with a will-message, aFleX prints the will-topic up to 1 MB only.
  - If the field or flag does not exist, then the output value is -1, so that users can detect the situation.

For information about aFleX commands, see aFleX Commands.

For information about MQTT events, see MQTT Events.

MQTT::clean_session_flag

Description Gets the flag for an MQTT CONNECT message.

Syntax

```
MQTT::clean_session_flag
```

Example Uses the following example to log the clean session flag value.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::clean_session_flag]"
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::client_id

Description Gets the client identifier for an MQTT CONNECT message.

Syntax

```
MQTT::client_id
```

Example Uses the following example to log the client id.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::client_id]"
                }
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::collect

Description Collects at least bytes of payload

Syntax

```
MQTT::collect
```

Collect the entire payload of the MQTT message or the maximum length of the payload (which is 1MB). If the message is longer than 1MB, the processing can only be done on the first 1MB.

```
MQTT::collect <size>
```

Collect size of payload of the MQTT message. The maximum length of data that can be collected is 1MB. The collected data can be accessed via the MQTT::payload command.

Example Use the following example to collect at least bytes of payload.

```
when MQTT_CLIENT_MESSAGE {
    MQTT::collect
}

when MQTT_CLIENT_MESSAGE_DATA {
    if { [MQTT::type] equals 8} {
        log "payload in PUBLISH is [MQTT::payload]"
    }
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_SERVER_MESSAGE

MQTT::drop

Description Drop the current MQTT message

Syntax

```
MQTT::drop
```

Example Use the following example to drop the MQTT message from the server side which includes "5min" in its topic:

```
when MQTT_SERVER_MESSAGE {
    if { [MQTT::topic] contains "5min" } {
        MQTT::drop
    }
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::dup_flag

Description Gets the duplicate flag for an MQTT PUBLISH message.

Syntax

```
MQTT::dup_flag
```

Example Use the following example to log the dup flag value.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::dup_flag]"
        }
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::keep_alive

Description Gets the keep_alive field for an MQTT CONNECT message.

Syntax

```
MQTT::keep_alive
```

Example Use the following example to log the keep_alive value.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::keep_alive]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE

- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::length

Description Gets the length for an MQTT message.

Syntax

```
MQTT::length
```

Example Use the following example to log the length for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::length]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::packet_id

Description Gets the packet-id for an MQTT message and sets the packet-id of the MQTT message to the given value and the value range is [0 to 65535].

Syntax

```
MQTT::packet_id
MQTT::packet_id <packet-id>
```

Example Use the following examples to log the packet Id for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::packet_id]"
}
```

```
when MQTT_CLIENT_MESSAGE {
    MQTT::packet_id -1
```

```
    MQTT::packet_id 1 1
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::password

Description Gets the password field for an MQTT CONNECT message.

Syntax

```
MQTT:password
```

Example Use the following example to log a password.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::password]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::payload

Description Gets the payload of an MQTT PUBLISH message

Syntax

```
MQTT::payload
```

Gets the payload of an MQTT PUBLISH message

```
MQTT::payload [<replace | prepend | append> <content>]
```

`MQTT::payload replace <content>` - Replace the entire payload of an MQTT PUBLISH message with a specific content

`MQTT::payload replace <offset> <size> <content>` - Replace the

payload starting from offset to offset + size with the given content
`MQTT::payload prepend <content>` - Add the given content to the beginning of the payload
`MQTT::payload append <content>` - Add the given content to the end of the payload

Example Use the following example to log the payload of an MQTT PUBLISH message.

```
when MQTT_CLIENT_MESSAGE_DATA {
log "[MQTT::payload]"
}
```

Example Use the following example to log the payload of an MQTT replace, prepend, append message.

```
when MQTT_CLIENT_MESSAGE_DATA {
    MQTT::payload replace test
    MQTT::payload prepend test:
    MQTT::payload append :test
    log "[MQTT::payload]"
}
```

Example Use the following example to log the payload of an MQTT replace message.

```
when MQTT_CLIENT_MESSAGE_DATA {
MQTT::payload replace 5 10 aflex_data
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::payload_length

Description Gets the payload length for an MQTT PUBLISH message.

Syntax `MQTT::payload_length`

Example Use the following example to log the payload length for an MQTT PUBLISH message

```
when MQTT_CLIENT_MESSAGE_DATA {
        log "[MQTT::payload_length]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::protocol_name

Description Gets the protocol name for an MQTT CONNECT message.

Syntax `MQTT::protocol_name`

Example Use the following example to log the protocol name for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::protocol_name]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::protocol_version

Description Gets the protocol review level for an MQTT CONNECT message.

Syntax

```
MQTT::protocol_version
```

Example

Use the following example to log the protocol version for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::protocol_version]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::qos

Description

Gets the Quality of Services (QoS) for an MQTT PUBLISH message. Supports the following three types of QoS:

- At most once
- At least once
- Exactly once

`MQTT::qos 0` - Set the QoS for MQTT PUBLISH message to 0. If the previous QoS is not 0, then the packet_id part will be removed in the output packet. The `MQTT::packet_id` can be called either right before or right after this command to get the original packet-id, if necessary.

`MQTT::qos 1`- Set the QoS of MQTT PUBLISH message to 1. If the previous QoS is 0, then the `MQTT::packet_id` must be called after this command to guarantee that MQTT protocol is strictly followed since the packet_id field is not included in MQTT PUBLISH message when QoS is 0.

`MQTT::qos 2` - Set the QoS of MQTT PUBLISH message to 2. If the previous QoS is 0, `MQTT::packet_id` must be called after this command to guarantee that MQTT protocol is strictly followed since the packet_id field is not included in MQTT PUBLISH message when QoS is 0.

Syntax

```
MQTT::qos

MQTT::qos <0 | 1 | 2>
```

Example Use the following examples to log the QoS for an MQTT message:

```
when MQTT_CLIENT_MESSAGE {
log "[MQTT::qos]"
}
```

```
when MQTT_PUBLISH {
set old_qos [MQTT::qos]
log "In MQTT_PUBLISH event, initial qos=[MQTT::qos], packet_
id=[MQTT::packet_id] "
if {$old_qos==0 } {
MQTT::packet_id [expr {int (rand()*65000)}]
}
MQTT::qos 2
log "After setting qos as 2, the new qos = [MQTT::qos],
packet_
id=[MQTT::packet_id] "
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::replace

Description This command replaces an MQTT message.

Syntax

```
MQTT::replace type <PUBACK | PUBREC | PUBREL | PUBCOMP |
UNSUBACK> packet_id <packet- id-number>
```

Example Use the following example to replace the current MQTT message.

```
when MQTT_CLIENT_MESSAGE {
```

```
        MQTT::replace type PUBACK packet_id 111
        MQTT::replace type PUBREC packet_id 111
        MQTT::replace type PUBREL packet_id 111
        MQTT::replace type PUBCOMP packet_id 111
        MQTT::replace type UNSUBACK packet_id 111
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::respond

Description This command transmits an MQTT message to sender of the incoming message.

Syntax

```
MQTT::respond type <PUBACK | PUBREC | PUBREL | PUBCOMP |
UNSUBACK> packet_id <packet-id-number>
```

Example Use the following example to transmit an MQTT message to sender.

```
when MQTT_CLIENT_MESSAGE {
        MQTT::respond type PUBACK packet_id 111
        MQTT::respond type PUBREC packet_id 111
        MQTT::respond type PUBREL packet_id 111
        MQTT::respond type PUBCOMP packet_id 111
        MQTT::respond type UNSUBACK packet_id 111
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::retain_flag

Description Gets the retain flag for an MQTT PUBLISH message

Syntax

```
MQTT::retain_flag
```

Gets the retain flag for an MQTT PUBLISH message

```
MQTT::retain_flag <0, 1>
```

Set the retain flag of MQTT PUBLISH messages

Example Use the following example to log the retain flag value for an MQTT message.

Example 1 - MQTT::retain_flag

```
when MQTT_CLIENT_MESSAGE_DATA {
log "[MQTT::retain_flag]"
}
```

Example Use the following example to get retain flag for an MQTT message.

```
when MQTT_SERVER_MESsAGE {
    MQTT::retain_flag 0
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::return_code

Description Gets the return-code field for an MQTT CONNACK message.

Syntax

```
MQTT::return_code
```

Example Use the following example to log the return code for an MQTT message.

```
when MQTT_SERVER_MESSAGE_DATA {
```

```
        log "[MQTT::return_code]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::return_code_list

Description Gets the return-code-list for multiple MQTT SUBACK messages.

Syntax

```
MQTT::return_code_list
```

Example Use the following example to log the return code list for multiple MQTT messages.

```
when MQTT_SERVER_MESSAGE_DATA {
        log "[MQTT::return_code_list]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::session_present_flag

Description Gets the session_present flag for an MQTT CONNACK message.

Syntax

```
MQTT::session_present_flag
```

Example Use the following example to log the value of session present flag for an MQTT message.

```
when MQTT_SERVER_MESSAGE_DATA {
```

```
        log "[MQTT::session_present_flag]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::topic

Description Get the list of the topic names from SUBSCRIBE/UNSUBSCRIBE message, or the topic name from PUBLISH message.

MQTT::topic add <topic-name> - Add a new topic to the given topic name in the topic list of SUBSCRIBE/UNSUBSCRIBE message. If QoS is given, the QoS value of the added topic will be set when it is inserted into SUBSCRIBE messages.
MQTT::topic count – Return the number of topics in the current message. PUBLISH message will have this equal to 1.
MQTT::topic delete first <topic-name> - Delete the first matched topic from MQTT SUBSCRIBE/UNSUBSCRIBE message.
MQTT::topic delete all <topic-name> - Delete all the matched topics from MQTT SUBSCRIBE/UNSUBSCRIBE message.

NOTE: When calling `MQTT::topic delete` onto MQTT SUBSCRIBE/UNSUBSCRIBE message, it is recommended that user calls `MQTT::topic count` to check whether the amount of topic is no less than 1, or the protocol violation will occur.

MQTT::topic index <index-number> - Get the topic name from the given index of SUBSCRIBE/UNSUBSCRIBE message.
MQTT::topic replace <topic-name> - Set the topic name of PUBLISH message as specified. Only works for PUBLISH messages.

Syntax

```
MQTT::topic
```

```
MQTT::topic [add <topic-name> | delete <topic-name> | count |
index <index-number> | <replace <topic-name>]
```

Example Use the following example to log the topic for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::topic]"
}
```

Example Use the following example to get topics from an MQTT PUBLISH message.

```
when MQTT_CLIENT_MESSAGE_DATA{
                MQTT::topic replace test300
}
when MQTT_SERVER_MESSAGE_DATA {
                log "[MQTT::topic count]"
    log "[MQTT::topic]"
    log "[MQTT::topic index 0]"
    log "[MQTT::topic qos test]"
    log "[MQTT::topic add test100 2]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::type

Description Gets the type for an MQTT message. The following are the message types:

- Reserved (0):
- CONNECT (1): When a client requests to connect to a server.
- CONNACK (2): When a server acknowledges the connection from a client.
- PUBLISH 3 (3): When the server publishes a message.

- PUBACK (4): When the server acknowledges the publishing for a message.
- PUBREC (5): When the server receives the message for publication. (Part 1 of Assured Delivery)
- PUBREL (6): When the server releases the message for publication. (Part 2 of Assured Delivery)
- PUBCOMP (7): When the server completes the publication of message. (Part 3 of Assured Delivery)
- SUBSCRIBE (8): When a client subscribes to a request.
- SUBACK (9): When a server acknowledges the subscription to a client.
- UNSUBSCRIBE (10): When a client unsubscribes a request.
- UNSUBACK (11): When a server unsubscribes an acknowledgment.
- PINGREQ (12): When a client pings a request to the server.
- PINGRESP (13): When the server pings a response to the client.
- DISCONNECT (14):When the client is disconnected with the server.
- Reserved (15):

Syntax

```
MQTT::type
```

Example Use the following example to log the type for an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::type]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::username

Description Gets the username field for an MQTT CONNECT message.

Syntax

```
MQTT::username
```

Example

Use the following example to log the username included in an MQTT message.

```
when MQTT_CLIENT_MESSAGE {
        log "[MQTT::username]"
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

MQTT::will

Description

Gets and sets the parts of the will message for an MQTT Connect message in the following sequence:

- will-topic
- will-message
- will-qos
- will-retain flag
- will-flag

MQTT::will - Manipulates the will-topic, will-message, will-qos and will-retain fields of MQTT CONNECT message
MQTT::will will-topic - Get will-topic field of CONNECT message
MQTT::will will-topic <will-topic> - Set the will-topic field to the given string
MQTT::will will-message - Get the will-message field of CONNECT message
MQTT::will will-message <will-message> - Set the will-message field to the given string
MQTT::will will-qos - Get the will-qos value of CONNECT message
MQTT::will will-qos <will-qos> - Set the will-qos value to be the given value. The given value must reside within [0, 2].
MQTT::will will-retain-flag - Get the will-retain field of CONNECT

message

MQTT::will will-retain-flag <will-retain> - Set the will-retain value to be the given value. The given value must reside within [0, 1]

Alternatively, you can create commands such as "`MQTT::will will-message`", "`MQTT::will will-message will-qos will-retain-flag`" , and even as "`MQTT::will will-message will-message`". These commands print the applicable fields in the defined sequence.

NOTE: Since `MQTT::will` now supports set methods, we no longer support commands like `MQTT::will will-topic will-message` (which has three input parameters) to obtain will-topic and will-message sequentially.

Syntax

```
MQTT::will [will-topic [<will-topic>] | will-message [<will-message>] | will-qos [<will-qos>] | will-retain-flag [<will-retain-flag>]]
```

Example Use the following example to get the will message for an MQTT Connect message.

```
when MQTT_CLIENT_MESSAGE {
        MQTT::will will-topic aflexTest
        MQTT::will will-message aflexTest
        MQTT::will will-qos 0
        MQTT::will will-retain-flag 0
}
```

Valid Events

- MQTT_CLIENT_MESSAGE
- MQTT_CLIENT_MESSAGE_DATA
- MQTT_SERVER_MESSAGE
- MQTT_SERVER_MESSAGE_DATA

Operation Commands

The following commands related to configuration are supported:

- OPER::get
- OPER::set

For information about aFleX commands, see aFleX Commands.

OPER::get

Description This command gets the current operational value of a specified system parameter or context-specific variable.

Syntax

```
OPER::get <parameter>
```

Example Use the following example to retrieve and log the current CPU usage when a client connection is accepted:

```
when CLIENT_ACCEPTED {
    set cpu_usage [OPER::get cpu-usage]
    log "Current CPU usage: $cpu_usage%"
}
```

OPER::set

Description This command sets or updates the value of a user-defined operational variable at runtime.

Syntax

```
OPER::set <parameter> <value>
```

Example Use the following example to set the operational variable `user_role` to `admin` when an HTTP request is received, and log its value.

```
when HTTP_REQUEST {
    OPER::set user_role "admin"
    log "User role set to [OPER::get user_role]"
}
```

Policy-Based SLB Commands

The following Policy-Based SLB command is supported:

- POLICY::bwlist id
- POLICY::source_rule

For information about aFleX commands, see aFleX Commands.

POLICY::bwlist id

Description Returns the group ID associated with an IP address in a black/white list.

Syntax

```
POLICY::bwlist id <ip>
```

This command causes the ACOS device to look in the black/white list that is bound to the same virtual port to which the aFleX policy is bound.

```
POLICY::bwlist id <ip> <bwlist_name>
```

This command causes the ACOS device to look in the specified list.

NOTE: When using `POLICY::bwlist` without a file name, the virtual port requires a Policy Template with Black-White List file.

Example Use the following example to black/white list an IP address.

```
when HTTP_REQUEST {
    if { [POLICY::bwlist id [IP::client_addr]] == 10 } {
        pool sg-internal
    } elseif { [POLICY::bwlist id [IP::client_addr] bwfile] ==
20 } {
        pool sg-www
    } else {
        reject
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

POLICY::source_rule

Description This command specifies the source rule name in the policy template which is to be used during the policy matching process. Each source rule has its priority. Even though the aFleX script selects a source rule, the priority of this rule is used to compare with the source rule selected by the original policy template matching (without aFleX). The higher priority rule is chosen.

Syntax

```
POLICY:: source_rule set <source destination match rule name>
```

Example Use the following example to set the source destination match rule to policy source matching.

```
when HTTP_REQUEST {
if { [HTTP::header exists "PASS"] } {
log "Header is matched. Set EP source rule as \"source-1\""
POLICY::source_rule set source-1
}
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA
- HTTP_RESPONSE_CONTINUE

RADIUS Message Load-balancing Commands

The following commands are supported for RADIUS message load-balancing:

- RADIUS::avp
- RADIUS::code
- RADIUS::id
- RADIUS::length

For information about aFleX commands, see aFleX Commands.

RADIUS::avp

Description This command returns attribute-value pairs (AVPs) from a RADIUS message. Bind the virtual port that uses this aFleX command to UDP port 1812. It supports both IPv4 and IPv6 AVPs.

Syntax

```
RADIUS::avp [<attr>]
```

The `<attr>` option specifies a RADIUS attribute, 1-255 (RFC 2865).
If an `<attr>` option is specified, it returns a list containing only the values that match the attribute number.
If an `<attr>` option is not specified, it returns a list of tuples. Each tuple contains the attribute number, length, and the corresponding value {attr, len, value}.

NOTE: The RADIUS AVP 40 contains the start or stop message.

Example Use the following example to return attribute-value pairs (AVPs) from a RADIUS message.

Consider the following list of AVPs:

```
set avpList {
    {1 "Hello"}
    {2 "World"}
    {3 "01234"}
}
```

To return the attribute numbers and their corresponding values for all AVPs present in the RADIUS message and print their values to the console, use the following example:

```
when CLIENT_DATA {
    set avpList [RADIUS::avp]
    foreach avpTuple $avpList {
        log "Attribute: [lindex $avpTuple 0], Value: [lindex
$avpTuple 2]"
    }
```

Output

```
Attribute: 1, Value: Hello
Attribute: 2, Value: World
Attribute: 3, Value: 01234
```

To return AVPs for a specific RADIUS attribute (here, attribute number 2), use the following example:

```
when CLIENT_DATA {
    set attrNumber 2
    set avpList [RADIUS::avp $attrNumber]
    foreach attrValue $avpList {
        log "Value for attribute $attrNumber: $attrValue"
    }
}
```

Output

```
Value for attribute 2: World
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

RADIUS::code

Description This command returns the Code field of a RADIUS message.

Syntax

```
RADIUS::code
```

Example Use the following example to log the code field of a RADIUS message.

```
when CLIENT_DATA {
        log "RADIUS Code: [RADIUS::code]"
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

RADIUS::id

Description This command returns the Identifier field of a RADIUS message.

Syntax

```
RADIUS::id
```

Example Use the following example to log the Identifier field of a RADIUS message.

```
when CLIENT_DATA {
        log "RADIUS Identifier: [RADIUS::id]"
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

RADIUS::length

Description This command returns the Length field of a RADIUS message.

Syntax

```
RADIUS::length
```

Example Use the following example to log the Length field of a RADIUS message.

```
when CLIENT_DATA {
        log "RADIUS Length: [RADIUS::length]"
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

RAM Caching Commands

The following RAM caching commands are supported on HTTP traffic (original proxy) and HTTP2 traffic (new proxy):

- CACHE::disable
- CACHE::enable
- CACHE::expire
- CACHE::hits

These commands are supported on HTTP traffic (the original proxy), but not supported on HTTP2 traffic (the new proxy).

- CACHE::age
- CACHE::headers

For information about aFleX commands, see aFleX Commands.

For information about RAM caching events, see RAM Caching Events.

CACHE::age

Description This command returns the age (in seconds) of a cached object. The age is how long the object has been in the cache.

Syntax

```
CACHE::age
```

Example Use the following example to return the age of a cached object in 60 seconds.

```
when CACHE_REQUEST {
    if { [CACHE::age] > 60 } {
        CACHE::expire
        log "The cache content expires when age > 60 seconds"
    }
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST
- HTTP_RESPONSE

CACHE::disable

Description This command disables the current HTTP request from being cached.

NOTE: The HTTP_RESPONSE_DATA event is not supported on the HTTP2 (new proxy).

Syntax

```
CACHE::disable
```

Example Use the following example to disable the current HTTP request from being cached.

```
when HTTP_REQUEST {
  switch -glob [HTTP::uri] {
    "*.jpg" { CACHE::enable }
```

```
    "*.png" { CACHE::enable }
    "*.gif" { CACHE::enable }
    "*.css" { CACHE::enable 86400 }
    "*.js" { CACHE::enable 86400 }
    default { CACHE::disable }
  }
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST
- HTTP_RESPONSE

CACHE::enable

Description This command caches an object, with the possibility of specifying how long to cache the object for.

Syntax

```
CACHE::enable [<age>]
```

The <age> option specifies how long the object should be cached for, in seconds. If the <age> option is not used, then the default time is the age in the RAM caching template.

Example Use the following example to enable the current HTTP request from being cached.

```
when HTTP_REQUEST {
  switch -glob [HTTP::uri] {
    "*.jpg" { CACHE::enable }
    "*.png" { CACHE::enable }
    "*.gif" { CACHE::enable }
    "*.css" { CACHE::enable 86400 }
    "*.js" { CACHE::enable 86400 }
    default { CACHE::disable }
  }
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

CACHE::expire

Description This command removes an object from the cache. It must be revalidated by the server to be cached again.

Syntax

```
CACHE::expire
```

Example Use the following example to remove the cached object to be revalidated from the server.

```
when RULE_INIT {
    set ::expirecache 0
}
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/expirecache" } {
        set ::expirecache 1
    }
}
when CACHE_RESPONSE {
    if { $::expirecache == 1 } {
        CACHE::expire
        log "Cache must be revalidated by [IP::client_addr]"
        set ::expirecache 0
    }
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST
- HTTP_RESPONSE

CACHE::headers

Description This command returns the HTTP headers of a cached object. The name and value of header fields are returned in a Tcl list.

Syntax

```
CACHE::headers
```

Example Use the following example to return the HTTP headers of a cached object.

```
when CACHE_RESPONSE {
    log "Cache Headers: [CACHE::headers]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST
- HTTP_RESPONSE

CACHE::hits

Description This command returns the number of hits in the cache for a cached object.

Syntax

```
CACHE::hits
```

Example The following example logs the number of cache hits for a specific `HTTP::uri`:

```
when HTTP_REQUEST {
    log "CACHE Hits: There are [CACHE::hits] hits for
[HTTP::uri]"
}
```

Valid Events

- CACHE_REQUEST
- CACHE_RESPONSE
- HTTP_REQUEST

- HTTP_RESPONSE

Resolve Commands

The following DNS resolution command is supported:

RESOLVE::lookup

For information about aFleX commands, see aFleX Commands.

RESOLVE::lookup

Description This command sends a DNS request to the DNS server for the list of IP addresses associated with the specified domain name. This command works when the DNS server is in asynchronous mode.

Syntax

```
RESOLVE::lookup <domain_name>
```

This command performs a DNS lookup for the specified domain name, using the default DNS server.

NOTE: Using the following CLI command, configure a primary DNS server: ip dns primary ip_address. A secondary DNS server can be configured using the following CLI command: ip dns secondary ip_address. This command will use the default DNS server. In case the default DNS server fails, if a secondary DNS server is configured, the command `RESOLVE::lookup` will use the secondary DNS server.

```
RESOLVE::lookup <server> <domain_name>
```

This command performs a DNS lookup for the specified domain name, using the specified DNS server.

NOTE: For HTTP or HTTPS virtual ports, the valid events are HTTP_REQUEST and HTTP_REQUEST_DATA. For TCP-proxy, the valid events are CLIENT_ACCEPTED and CLIENT_DATA.

This command is only supported for use with IPv4 addresses.

This command is not supported in L3V partitions.

Use the following example to perform a DNS lookup with the default DNS server:

```
when HTTP_REQUEST {
    log "RESOLVE Lookup: [HTTP::host] resolves to
[RESOLVE::lookup [HTTP::host]]"
}
```

Use the following example to perform a DNS lookup with the specified DNS server:

```
when HTTP_REQUEST {
    log "RESOLVE Lookup: [HTTP::host] resolves to
[RESOLVE::lookup @8.8.8.8 [HTTP::host]]"
}
```

Use the following example to dynamically choose the DNS server for the DNS lookup, and then perform the DNS lookup:

```
when HTTP_REQUEST {
        set cnt 0
        set s1 192.168.1.1
        set s2 192.168.1.2
        set client_ip [IP::client_addr]
        set method [HTTP::method]
        set uri [HTTP::uri]
        log "client ip = $client_ip"
        if {[expr $cnt % 2]} {
                set server "$s1"
        } else {
                set server "$s2"
        }
        set ips [RESOLVE::lookup @$server "www.example.com"]
        log "cnt = $cnt  server = '$server' ips = '$ips'"
        log "HTTP method = '$method' uri = '$uri'"
        incr $cnt 1
}
when HTTP_RESPONSE {
        log "Response: HTTP method = '$method' uri = '$uri'"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- CLIENT_ACCEPTED
- CLIENT_DATA

SIP Commands

The following SIP commands are supported:

- SIP::call_id
- SIP::from
- SIP::header
- SIP::method
- SIP::respond
- SIP::response
- SIP::to
- SIP::uri
- SIP::via

For examples of the SIP command in use, see SIP Command Examples.

For information about aFleX commands, see aFleX Commands.

For information about SIP events, see SIP Events.

SIP::call_id

Description This command returns the value of the Call-ID header in a SIP request.

Syntax

```
SIP::call_id
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::from

Description This command returns the value of the “From” header in a SIP request.

Syntax

```
SIP::from
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED

- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::header

Description This command either returns the specified SIP header, or else it inserts a header name and a corresponding header value into the SIP header.

Syntax

```
SIP::header [<value>] <header-name> [<index>]
```

The `<value>` option specifies the header value. The `<index>` option specifies which header level the value applies to in case of multiple header levels. If an index is not specified, then aFleX applies the value to the first header corresponding to the header-name.

```
SIP::header insert <header-name> <header-value> [<index>]
```

The `<index>` option specifies where to insert the new header. If the optional index does not exist, then a "via" header is inserted at the beginning of the SIP headers, and all other headers are inserted at the end of the SIP headers. If the index is not specified, then the header is inserted before other headers with the same name and value.

```
SIP::header remove <header-name> [index]
```

Remove the `<header-name>` in the SIP header. The `<index>` option specifies which header is applied.

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED

- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::method

Description This command returns what type the SIP request method is.

Syntax

```
SIP::method
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::respond

Description This commands returns a response with the defined code, phrase, and header name and corresponding header value.

Syntax

```
SIP::respond code <"phrase" <"header-name" "header-value">>
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::response

Description This command returns the SIP response code or phrase. You can also use this command to rewrite the response code or phrase.

Syntax

```
SIP::response code
```

The above command returns the SIP response code.

```
SIP::response phrase
```

The above command returns the SIP response phrase.

```
SIP::response rewrite code <phrase>
```

The above command rewrites the response code or phrase.

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::to

Description This command returns the value of the “To” header in the SIP request.

Syntax

```
SIP::to
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::uri

Description This command returns the request’s URI.

Syntax

```
SIP::uri
```

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST

- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP::via

Description This command returns the “via” information for SIP.

Syntax

```
SIP::via [<index>]
```

The above command returns the information in the SIP “via” header. If the `<index>` option is specified, then only the information at that index is returned.

```
SIP::via proto [<index>]
```

The above command returns the SIP “via” protocol part. If the `<index>` option is specified, then only the information at the index is returned.

```
SIP::via sent_by [<index>]
```

The above command returns the “sent by” from the SIP “via” information. If the `<index>` option is specified, then only the information at the index is returned.

```
SIP::via received [<index>]
```

The above command returns the “received” value of the SIP “via” information. If the `<index>` option is specified, then only the information at the index is returned.

```
SIP::via branch [<index>]
```

The above command returns the “branch” value of the SIP “via” information. If the <index> option is specified, then only the information at the index is returned.

```
SIP::via maddr [<index>]
```

The above command returns the multicast address value of the SIP “via” information. If the <index> option is specified, then only the information at the index is returned.

```
SIP::via ttl [<index>]
```

The above command returns the TTL value of the SIP “via” information. If the <index> option is specified, then only the information at the index is returned.

Example See SIP Command Examples.

Valid Events

- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Valid with the following IP, TCP and UDP events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

SIP Command Examples

This section contains three example scripts that incorporate SIP commands:

- Example 1
- Example 2
- Example 3

Example 1

Use the following example to log all the available header values when a full SIP request is received from the client and the SIP request method is subscribe.

```
when SIP_REQUEST {
        if { [SIP::method] contains "SUBSCRIBE" } {
        log "*****************  SIP-REQUEST  *******************"
        log "SIP::call_id is [SIP::call_id]"
        log "-------------------------------------------------"
        log "SIP::from is [SIP::from]"
        log "-------------------------------------------------"
        log "SIP::header Via [SIP::header Via]"
        log "SIP::header Via value index0 [SIP::header value Via 0]"
        log "SIP::header Via index9 [SIP::header Via 9]"
        log "SIP::header From [SIP::header From]"
        log "SIP::header value From index0 [SIP::header value From 0]"
        log "SIP::header From index9 <not exist> [SIP::header From 9]"
        log "SIP::header To [SIP::header To]"
        log "SIP::header To index0 [SIP::header To 0]"
        log "SIP::header value To index9 <not exist> [SIP::header value To 9]"
        log "SIP::header Call-ID [SIP::header Call-ID]"
        log "SIP::header value Call-ID index0 [SIP::header value Call-ID 0]"
        log "SIP::header value Call-ID index9 <not exist> [SIP::header value
Call-ID 9]"
        log "SIP::header CSeq [SIP::header CSeq]"
        log "SIP::header CSeq value index0 [SIP::header value CSeq 0]"
        log "SIP::header CSeq index9 <not exist> [SIP::header CSeq 9]"
        log "SIP::header Contact [SIP::header Contact]"
        log "SIP::header value Contact index0 [SIP::header value Contact 0]"
        log "SIP::header Contact index9 <not exist> [SIP::header Contact 9]"
        log "SIP::header Max-Forwards [SIP::header Max-Forwards]"
        log "SIP::header Event [SIP::header Event]"
        log "SIP::header User-Agent [SIP::header User-Agent]"
        log "SIP::header Expires [SIP::header Expires]"
        log "SIP::header Allow [SIP::header Allow]"
        log "SIP::header Accept [SIP::header Accept]"
        log "SIP::header Content-length [SIP::header Content-length]"
        log "SIP::header abc <not valid header> [SIP::header abc]"
        log "-------------------------------------------------"
        SIP::header remove Via
```

```
        log "SIP::header remove Via [SIP::header Via]"
        SIP::header remove From
        log "SIP::header remove From [SIP::header From]"
        log "-------------------------------------------------"

        log "SIP::header Via 0 (request) [SIP::header Via 0]"
        log "SIP::response code [SIP::response code]"

        SIP::header insert Via "SIP/10.0/UDP
ss.under.test.com:5070;maddr=3ffe:501:ffff:50::51;ttl=1;branch=z9hG4bK721e
418c4.1" 10

        SIP::header insert event "SIP/2.0/UDP
ss.under.test.com:5070;maddr=3ffe:501:ffff:50::51;ttl=1;branch=z9hG4bK721e
418c4.1;
received=3ffe:501:ffff:50::50" 1 #      log "Event 0 is [SIP::header
event]"

        SIP::header insert From "<sip:218@mysip.com>;tag=1043119751"
        log "SIP::header insert From index1 [SIP::header From]"

        log "SIP::header From [SIP::header From]"
        SIP::header insert Via "SIP/2.0/UDP
171.1.1.217:5060;rport;branch=z9hG4bk11229103"
        log "SIP::header insert Via [SIP::header Via]"
        log "SIP::header From(2) [SIP::header From]"
        log "SIP::header insert xyz index9 [SIP::header insert xyz "x y z" 9]"
        log "-------------------------------------------------"
        log "SIP::method [SIP::method]"
        log "-------------------------------------------------"
        SIP::respond 401 "no way" From "future"
        log "-------------------------------------------------"
        log "SIP::response [SIP::response code]"
        log "SIP::response phase [SIP::response phrase]"

        SIP::response rewrite 402 "no xxx"
        log "SIP::response rewrite code phrase [SIP::response code]"
        log "-------------------------------------------------"
        log "SIP::to [SIP::to]"
        log "-------------------------------------------------"
```

```
        log "SIP::uri [SIP::uri]"
        log "-------------------------------------------------"
        log "SIP::via [SIP::via]"
        log "SIP::via index0 [SIP::via 0]"
        log "SIP::via index9 [SIP::via 9]"
        log "SIP::via proto [SIP::via proto]"
        log "SIP::via proto index0 [SIP::via proto 0]"
        log "SIP::via proto index9 [SIP::via proto 9]"
        log "SIP::via sent_by [SIP::via sent_by]"
        log "SIP::via sent_by index0 [SIP::via sent_by 0]"
        log "SIP::via sent_by index9 [SIP::via sent_by 9]"
        log "SIP::via received [SIP::via received]"
        log "SIP::via received index0 [SIP::via received 0]"
        log "SIP::via received index9 [SIP::via received 9]"
        log "SIP::via branch [SIP::via branch]"
        log "SIP::via branch index0 [SIP::via branch 0]"
        log "SIP::via branch index9 [SIP::via branch 9]"
        log "SIP::via maddr [SIP::via maddr]"
        log "SIP::via maddr index0 [SIP::via maddr 0]"
        log "SIP::via maddr index9 [SIP::via maddr 9]"
        log "SIP::via ttl [SIP::via ttl]"
        log "SIP::via ttl index0 [SIP::via ttl 0]"
        log "SIP::via ttl index9 [SIP::via ttl 9]"
        }
}
```

Example 2

Use the following example to look for SIP response codes, rewrite the codes to customized messages, and log them when a full SIP response is received from the server.

```
when SIP_RESPONSE {
        if { [SIP::response code] equals "401" } {
                SIP::response rewrite 411 Phrase_Unauthorized
                log "SIP::response code [SIP::response code]"
                log "SIP::response phrase [SIP::response phrase]"
        }
        if { [SIP::response code] equals "501" } {
                SIP::response rewrite 511 Phrase_Not_Implemented
                log "SIP::response code [SIP::response code]"
```

```
                log "SIP::response phrase [SIP::response phrase]"
        }
        if { [SIP::response code] equals "200" } {
                SIP::response rewrite 210 okok
                log "SIP::response code [SIP::response code]"
                log "SIP::response phrase [SIP::response phrase]"
        }
}
```

Example 3

Use the following example to log all the available header values when a full SIP request is received from the client and the SIP request method is subscribe.

```
when SIP_REQUEST_SEND {
        if { [SIP::method] contains "SUBSCRIBE" } {
        log "*****************  SIP-REQUEST-SEND  *******************"
        log "SIP::header Via 1 (request_sent) [SIP::header Via 1]"

        log "SIP::call_id is [SIP::call_id]"
        log "-------------------------------------------------"
        log "SIP::from is [SIP::from]"
        log "-------------------------------------------------"
        log "SIP::header Via [SIP::header Via]"
        log "SIP::header Via value index0 [SIP::header value Via 0]"
        log "SIP::header Via index9 [SIP::header Via 9]"
        log "SIP::header From [SIP::header From]"
        log "SIP::header value From index0 [SIP::header value From 0]"
        log "SIP::header From index9 <not exist> [SIP::header From 9]"
        log "SIP::header To [SIP::header To]"
        log "SIP::header To index0 [SIP::header To 0]"
        log "SIP::header value To index9 <not exist> [SIP::header value To 9]"
        log "SIP::header Call-ID [SIP::header Call-ID]"
        log "SIP::header value Call-ID index0 [SIP::header value Call-ID 0]"
        log "SIP::header value Call-ID index9 <not exist> [SIP::header value
Call-ID 9]"
        log "SIP::header CSeq [SIP::header CSeq]"
        log "SIP::header CSeq value index0 [SIP::header value CSeq 0]"
        log "SIP::header CSeq index9 <not exist> [SIP::header CSeq 9]"
        log "SIP::header Contact [SIP::header Contact]"
        log "SIP::header value Contact index0 [SIP::header value Contact 0]"
```

```
        log "SIP::header Contact index9 <not exist> [SIP::header Contact 9]"
        log "SIP::header Max-Forwards [SIP::header Max-Forwards]"
        log "SIP::header Event [SIP::header Event]"
        log "SIP::header User-Agent [SIP::header User-Agent]"
        log "SIP::header Expires [SIP::header Expires]"
        log "SIP::header Allow [SIP::header Allow]"
        log "SIP::header Accept [SIP::header Accept]"
        log "SIP::header Content-length [SIP::header Content-length]"
        log "SIP::header abc <not valid header> [SIP::header abc]"
        log "----------------------------------------------------"
        SIP::header remove Via
        log "SIP::header remove Via [SIP::header Via]"

        SIP::header remove From
        log "SIP::header remove From [SIP::header From]"

        SIP::header remove From
        log "SIP::header remove From [SIP::header From]"

        SIP::header remove abc
        log "SIP::header remove index To [SIP::header abc]"

        log "----------------------------------------------------"
        SIP::header insert From "<sip:218@mysip.com>;tag=1043119751"
        log "SIP::header insert From index1 [SIP::header From]"

        log "SIP::header From [SIP::header From]"
        SIP::header insert Via "SIP/2.0/UDP 171.1.1.217:5060;rport;
branch=z9hG4bk11229103"
        log "SIP::header insert Via [SIP::header Via]"
        log "SIP::header From(2) [SIP::header From]"
        log "SIP::header insert xyz index9 [SIP::header insert xyz "x y z" 9]"
        log "----------------------------------------------------"
        log "SIP::method [SIP::method]"
        log "----------------------------------------------------"
        SIP::respond 401 "no way" From "future"
        log "----------------------------------------------------"
        log "SIP::response [SIP::response code]"
        log "SIP::response phase [SIP::response phrase]"
```

```
        SIP::response rewrite 402 "no xxx"
        log "SIP::response rewrite code phrase [SIP::response code]"
        log "-------------------------------------------------"
        log "SIP::to [SIP::to]"
        log "-------------------------------------------------"
        log "SIP::uri [SIP::uri]"
        log "-------------------------------------------------"
        log "SIP::via [SIP::via]"
        log "SIP::via index0 [SIP::via 0]"
        log "SIP::via index9 [SIP::via 9]"
        log "SIP::via proto [SIP::via proto]"
        log "SIP::via proto index0 [SIP::via proto 0]"
        log "SIP::via proto index9 [SIP::via proto 9]"
        log "SIP::via sent_by [SIP::via sent_by]"
        log "SIP::via sent_by index0 [SIP::via sent_by 0]"
        log "SIP::via sent_by index9 [SIP::via sent_by 9]"
        log "SIP::via received [SIP::via received]"
        log "SIP::via received index0 [SIP::via received 0]"
        log "SIP::via received index9 [SIP::via received 9]"
        log "SIP::via branch [SIP::via branch]"
        log "SIP::via branch index0 [SIP::via branch 0]"
        log "SIP::via branch index9 [SIP::via branch 9]"
        log "SIP::via maddr [SIP::via maddr]"
        log "SIP::via maddr index0 [SIP::via maddr 0]"
        log "SIP::via maddr index9 [SIP::via maddr 9]"
        log "SIP::via ttl [SIP::via ttl]"
        log "SIP::via ttl index0 [SIP::via ttl 0]"
        log "SIP::via ttl index9 [SIP::via ttl 9]"
        }
}
```

SMTP Commands

The following category commands is supported:

- SMTP::mail
- SMTP::greet
- SMTP::ehlo

For information about aFleX commands, see aFleX Commands.

For information about SMTP events, see SMTP Events

SMTP::mail

Description Retrieves MAIL command parameter (reverse-path).

NOTE: `Reverse-path` is the first parameter to follow whereas mail-param is optional.

Syntax

```
SMTP::mail [reverse-path | mail-param]
```

Example Use the following example to retrieve the MAIL command parameter.

```
When SMTP_MAIL {
If {[SMTP::mail] equals abc.com} {
                node 1.1.1.1 25
                        } else {
                                Node 2.2.2.2 25
                                }
                }
```

Valid Events

Valid with the following SMTP events:

- SMTP_MAIL

SMTP::greet

Description Set EHLO ok messages.

Syntax

```
SMTP::greet [messages]
```

Example Use the following example to set EHLO ok messages.

```
When SMTP_EHLO {
        SMTP::greet "VRFY"
        }
```

Valid Events

Valid with the following SMTP events:

- SMTP_EHLO

SMTP::ehlo

Description Retrieve client's ehlo/helo message.

Syntax

```
SMTP::ehlo
```

Example Use the following example to retrieve the client's ehlo message.

```
When SMTP_EHLO {
        SMTP::ehlo
        }
```

Valid Events

Valid with the following SMTP events:

- SMTP_EHLO

SSL Commands

The following SSL commands are supported:

- SSL::authenticate
- SSLI::bypass
- SSLI::cache_cert
- SSL::cert
- SSL::cipher
- SSL::collect
- SSL::disable
- SSLI::drop
- SSL::enable
- SSL::extensions
- SSL::hostname
- SSLI::inspect
- SSL::mode
- SSL::payload
- SSL::release
- SSL::renegotiate
- SSL::respond
- SSL::session invalidate
- SSL::sessionid
- SSL::template
- SSL::verify_result

For information about aFleX commands, see aFleX Commands.

For information about SSL events, see SSL Events.

SSL::authenticate

Description Use the following command to permanently, or for a single occurrence, authenticate client SSL certificates. To set the depth to which the authenticity of the certificate is inspected, use the keyword `depth` followed by a number.

Syntax

```
SSL::authenticate [once | always | depth <number>]
```

Example Use the following example to set the index and renegotiate variables when a client establishes a connection with the ACOS device. After the initial ssl handshake, the client authentication parameters will be changed using the `SSL::authenticate` command. `SSL::authenticate` will require the client to be authenticated once. `SSL::authenticate` depth 6 will verify the client certificate until depth 6. After this, when we renegotiate, if the certificate used for client authentication has depth more than 6, the handshake should fail. If the SSL handshake is successful the "SSL authenticate invalid CLIENTSSL_DATA: FAIL" message would be printed in the logs.

```
when CLIENT_ACCEPTED {
  set do_reneg 1
  set index 1
}
when CLIENTSSL_HANDSHAKE {
  SSL::collect
  if {$do_reneg} {
    log "Normal handshake for SSL authenticate invalid
CLIENTSSL_DATA"
    log "Index for SSL authenticate invalid CLIENTSSL_DATA:
$index"
    incr index
    set do_reneg 0
  } else {
    log "SSL authenticate is invalid CLIENTSSL_DATA: FAIL"
    log "Index for SSL authenticate invalid CLIENTSSL_DATA:
$index"
    incr index
  }
```

```
}
when CLIENTSSL_DATA {
     log "Start SSL authenticate invalid CLIENTSSL_DATA"
     SSL::authenticate once
     SSL::authenticate depth 6
     SSL::cert mode require
     SSL::renegotiate
     SSL::release
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

SSL::cert

Description Use the following command to view information on SSL certificates.

Syntax

```
SSL::cert <level>
```

Use this command to return the SSL certificate with the specified level in the certificate chain. Level 0 is the first level. This command will provide certificate information in DER format. In release 2.6.1-P2 or earlier, this command will provide certificate information in text format.

When used with server-side events like `HTTP_REQUEST_SEND`, the `SSL::cert` command will fetch the server-side SSL certificate. If this operation fails, the aFlex script will abort. In releases earlier than 2.7.2, `SSL::cert` would incorrectly fetch the client-side certificate even for server-side events.

```
SSL::cert count
```

Use this command to return the number of certificates in the certificate chain.

```
SSL::cert issuer <index>
```

Use this command to return the issuer of the certificate with the specified level.

```
SSL::cert mode [request | require | ignore | auto]
```

Use this command to set the certificate mode. This setting will override the mode that is set in template.

NOTE: To specify an alternate format for the certificate, use X509::text to receive the certificate in text format, or X509::whole to receive the certificate in PEM format.
The commands `SSL::cert count` and `SSL::cert issuer` are not supported when used with CLIENTSSL_CLIENTHELLO event.

Example Use the following example to log the client certificate at level 0. X509::text is used to convert the binary into ASCII for verification.

```
when CLIENTSSL_HANDSHAKE {
        log "SSL cert for CLIENTSSL_HANDSHAKE is [X509::text
[SSL::cert 0]]"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

SSL::cipher

Description Use the following command to return information on SSL ciphers.

Syntax

```
SSL::cipher name
```

Leverage the format of the OpenSSL SSL_CIPHER_get_name() function (for example, “EDH-RSA-DES-CBC3-SHA" or "RC4-MD5”) with this command to return the current SSL cipher name.

```
SSL::cipher version
```

Leverage the format of the OpenSSL SSL_CIPHER_get_version() function (for example, “SSLv2”, “SSLv3”, or “TLSv1”) with this command to return the current SSL cipher version.

```
SSL::cipher bits
```

Leverage the format of the OpenSSL SSL_CIPHER_get_bits() function (for example, 128 or 40) with this command to return the number of secret bits that the current SSL cipher uses.

Example

Use the following example to log the cipher name, cipher bits, and cipher version used in the SSL handshake.

```
when CLIENTSSL_HANDSHAKE {
        log "SSL cipher_name is [SSL::cipher name]"
        log "SSL cipher_bit is [SSL::cipher bits]"
        log "SSL cipher_version is [SSL::cipher version]"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

SSL::collect

Description

Use the following command to collect SSL application data.

Syntax

```
SSL::collect
```

Example Use the following example to collect the SSL application information when the client SSL handshake completes.

```
when CLIENTSSL_HANDSHAKE {
    SSL::collect
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_HANDSHAKE
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERHELLO

SSL::disable

Description Use the following command to turn off server or client SSL.

Syntax

```
SSL::disable [clientside | serverside]
```

NOTE: This command is only supported on the HTTP and the HTTPS. Other types are not supported.

Example Use the following example to disable SSL and server-side SSL.

```
when CLIENT_ACCEPTED {
        SSL::disable
        SSL::disable serverside
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_DATA

- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

SSL::enable

Description Use the following command to turn on client or server SSL.

NOTE: This command is only supported on the HTTP and the HTTPS. Other types are not supported.

Syntax

```
SSL::enable [clientside | serverside]
```

Example Use the following example to enable SSL and server-side SSL.

```
when CLIENT_ACCEPTED {
        SSL::enable
        SSL::enable serverside
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

SSL::extensions

Description Use the following command to parse the SSL certificate extensions.

Syntax

```
SSL::extensions count
```

Returns the number of SSL certificate extensions received.

```
SSL::extensions -index <extension_number>
```

Returns the byte array for the specified SSL certificate extension.

```
SSL::extensions -type <extension_type>
```

Returns the byte array for the specified SSL certificate extension type, or an empty string if not found. Returns only the first instance if the same extension type is present more than once.

```
SSL::extensions exists -type <extension_type>
```

Returns the following values:

0: No SSL certificate extension of the type is returned.

1: There is at least one SSL certificate extension of the type returned.

Valid Events

- CLIENTSSL_CLIENTHELLO
- SERVERSSL_SERVERHELLO
- SERVERSSL_CLIENTHELLO_SEND

SSL::hostname

Description Gets the host name from the header of the CLIENT_HELLO message. If the host name does not exist, it returns as NULL value.

Syntax

```
SSL::hostname [payload]
```

The payload refers to the byte array collected from the TCP stream that contains the CLIENT_HELLO message. It is used to extract data and return the hostname.
When payload is passed to SSL::hostname, the byte array of the CLIENT_HELLO message is provided to the command. The SSL::hostname command then parses this data to extract the hostname.

Example The following example extracts and logs the SSL hostname from incoming TCP packets:

```
when CLIENT_ACCEPTED {
                TCP::collect
}
```

```
when CLIENT_DATA {
                Log "[SSL::hostname]"
}
```

Example

The following example calculates CLIENT_HELLO length based on the SSL header. It collects data until the payload length exceeds the calculated CLIENT_HELLO length. Once the required amount of data is collected, it passes the collected TCP payload to SSL::hostname. This is particularly useful when the CLIENT_HELLO message is fragmented across multiple TCP packets.

```
when CLIENT_ACCEPTED {
TCP::collect
set packet_count 0
}
when CLIENT_DATA {
  log "packet count $packet_count"
  set offset 6
  set payload [TCP::payload]
  binary scan $payload @${offset}H6 length_hex
  set length [format %d 0x$length_hex]
  log "Rcv'd len [TCP::payload length]"
  log $length
  if {[TCP::payload length] < $length} {
  incr packet_count
  } else {
      log "CLient data ssl hostname is [SSL::hostname
$payload]" <--- can also use [SSL::hostname [TCP::payload]]
      TCP::release
   }
}
```

Valid Events

- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_HANDSHAKE
- CLIENT_DATA

SSL::mode

Description Use the following command on the server or the client to discover whether SSL has been enabled or disabled. This command will return 1, if SSL is turned on, or 2, if SSL is turned off.

Syntax

```
SSL::mode
```

NOTE: When the certificate mode is set with this command, it will override the mode set in the SSL template.

Example Use the following example SSL::mode command to ignore the example-client-ssl-template mode.

```
when CLIENT_ACCEPTED {
    SSL::template example-client-ssl-template
}
when HTTP_REQUEST {
    log "The SSL mode is [SSL::mode]."
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- SERVER_CONNECTED
- SERVERSSL_HANDSHAKE

SSL::payload

Description Use this command to return SSL data that has been collected, or to replace the collected payload with the information that is provided.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
SSL::payload <size>
```

Use this command to return the SSL content that has been collected.

```
SSL::payload <offset> <size>
```

Use this command to return the accumulated SSL content starting from `<offset>`.

```
SSL::payload <length>
```

Use this command to return the collected SSL data in bytes.

```
SSL::payload replace offset <length> <data>
```

Use this command to return the collected payload with the given data.

Example Use the following example to log the length of the SSL payload.

```
when CLIENTSSL_CLIENTCERT {
        SSL::collect 100
        log "Start collecting SSL data"
}
when CLIENTSSL_DATA {
        log  "SSL payload length is CLIENTSSL_CLIENTCERT:
[SSL::payload length]"
}
```

Example Use the following example to capture the original SSL payload. Replace the GET response of the request with data to get new payload. Replace the 0 to 50 bytes of SSL payload with the new payload.

```
when CLIENTSSL_CLIENTCERT {
        SSL::collect 100
        log "Start collecting SSL data"
}
when CLIENTSSL_DATA {
        set data [SSL::payload]
        set len [SSL::payload length]
        log "SSL payload length before replace = $len"
        log "SSL payload data before replace = $data"
```

```
        set new_payload [string map {GET ""} "data"]
        SSL::payload replace 0 50 "$new_payload"
        log "SSL payload data after replace is [SSL::payload]"
        log "SSL payload length after replace is [SSL::payload
length]"
}
```

Example Use the following example to log the SSL payload until size 100 of the total size.

```
when CLIENTSSL_CLIENTCERT {
        SSL::collect 164
}
when CLIENTSSL_DATA {
        log "SSL payload of size CLIENTSSL_CLIENTCERT is
[SSL::payload 100]"
}
```

Valid Events

- CLIENTSSL_DATA
- SERVERSSL_DATA

SSL::release

Description Use the following command to release the SSL collect mode. This will stop SSL application information from being gathered.

Syntax

```
SSL::release
```

Example Use the following example to release the data collected so that the session can continue. If the release fails, then the session fails.

```
when CLIENTSSL_HANDSHAKE {
    SSL::collect 120
}
when CLIENTSSL_CLIENTCERT {
        SSL::collect 100
        log "Start collecting SSL data"
}
when CLIENTSSL_DATA {
```

```
        log  "SSL payload length CLIENTSSL_CLIENTCERT:
[SSL::payload length]"
        SSL::release
}
```

Valid Events

- CLIENTSSL_DATA
- SERVERSSL_DATA

SSL::renegotiate

Description Only supported on devices with SSL Hardware. Use the following command on the client after the SSL handshake has been completed to mandate SSL renegotiation. Specify the `disable` keyword to prevent client-side SSL renegotiation.

NOTE: This command is not compatible with TLS 1.3, as the SSL renegotiation feature has been entirely removed from the TLS 1.3 protocol. As a result, this command has no effect when TLS 1.3 is in use.

Syntax

```
SSL::renegotiate [disable]
```

Example Use the following example to set the index and renegotiate variables when a client establishes a connection with the ACOS device. After the initial ssl handshake, the client authentication parameters will be changed using the SSL::authenticate command. SSL::authenticate once will require the client to be authenticated once. SSL::authenticate depth 6 will verify the client certificate until depth 6. After this, when we renegotiate, if the certificate used for client authentication has depth more than 6, the handshake should fail. If the SSL handshake is successful the "SSL authenticate invalid CLIENTSSL_DATA: FAIL" message would be printed in the logs.

```
when CLIENT_ACCEPTED {
  set do_reneg 1
  set index 1
}
when CLIENTSSL_HANDSHAKE {
  SSL::collect
```

```
  if {$do_reneg} {
    log "Normal handshake for SSL authenticate invalid
CLIENTSSL_DATA"
    log "Index for SSL authenticate invalid CLIENTSSL_DATA:
$index"
    incr index
    set do_reneg 0
  } else {
    log "SSL authenticate is invalid CLIENTSSL_DATA: FAIL"
    log "Index for SSL authenticate invalid CLIENTSSL_DATA:
$index"
    incr index
  }
}
when CLIENTSSL_DATA {
     log "Start SSL authenticate invalid CLIENTSSL_DATA"
     SSL::authenticate once
     SSL::authenticate depth 6
     SSL::cert mode require
     SSL::renegotiate
     SSL::release
}
```

Valid Events

- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

SSL::respond

Description This command is used to send specific SSL data to the client in a client-side event, or to the server in a server-side event. This command is supported on HTTP2 (new proxy).

NOTE:

- This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).
- This command is supported on HTTP2 (new proxy).

Syntax

```
SSL::respond <data>
```

Example

Use the following example to send your own response to the client instead of the server response.

```
when SERVERSSL_HANDSHAKE {
     SSL::collect 100
     log "Start collecting SSL data"
}
when SERVERSSL_DATA {
  set data [SSL::payload]
  if {$data contains "invite"} {
       SSL::respond "HTTP/1.1 200 OK\r\n\r\n Session
active.\r\n"
  } else {
      SSL::respond "HTTP/1.1 404 OK\r\n\r\n Session page not
found.\r\n"
  }
log "Sent SSL respond"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERHELLO

SSL::session invalidate

Description Use the following command after the SSL handshake to the client has been completed to ensure that the same SSL Session ID for the client is not used again.

Syntax

```
SSL::session invalidate
```

Example Use the following example to invalidate the current SSL session so the session id cannot be reused.

```
when HTTP_REQUEST {
        log "ssl session invalidate for HTTP_REQUEST is :
[SSL::session invalidate]"
}
```

Valid Events

- CLIENTSSL_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

SSL::session

Description This command retrieves session-related information for an ongoing SSL connection.

Syntax

```
SSL::session
```

Example Use the following example to log the current SSL session details when an HTTP request is received over an SSL connection:

```
when HTTP_REQUEST {
        log "SSL session information: [SSL::session]"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

SSL::sessionid

Description Returns the current SSL session ID for the client side only, not for the server side.

Syntax

```
SSL::sessionid
```

Example Use the following example to log the SSL session id generated during the SSL handshake.

```
when HTTP_REQUEST {
        log "SSL session id for current session is
[SSL::sessionid]"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

SSL::sessionsecret

Description Use the following command to return the SSL session key information during the SSL/TLS handshake when a client is establishing a connection with the server.

Syntax

```
SSL::sessionsecret
```

Example Use the following example to log the SSL session secret for the current SSL/TLS session.

```
when CLIENTSSL_HANDSHAKE {
     log "SSL session secrets for current session is
[SSL::sessionsecret]"
 }
```

The SSL session secrets are logged in the HEX string format: **<label> <client random> <secret>**.

NOTE: This API will only work if the **session-key-logging-enable** configuration option is enabled in the **ssl-client** and **ssl-server** templates.

Valid Events

- CLIENTSSL_HANDSHAKE
- SERVERSSL_HANDSHAKE

SSL::template

Description Use the following command on the client or the server connection to apply an SSL template.

Syntax

```
SSL::template <name>
```

Based on a client or server side, this command will apply the specified SSL template.

```
SSL::template [clientside|serverside] <name>
```

NOTE: This command is only supported on the HTTP and the HTTPS. Other types are not supported.

Example Use the following example to apply templates when a client establishes a connection.

```
when CLIENT_ACCEPTED {
    SSL::template example-client-ssl-template
    SSL::template serverside example-server-ssl-template
}
```

Valid Events

The following event is valid at client-side:

- CLIENT_ACCEPTED

The following events are valid at server-side:

- CLIENT_ACCEPTED
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- LB_SELECTED
- SERVER_CONNECTED
- SERVERSSL_HANDSHAKE

SSL::verify_result

Description Use the following command to either set the `<result_code>` for the peer certification verification or retrieve the result code of the peer certification verification.

Syntax

```
SSL::verify_result [<result_code>]
```

Example Use the following example to log the SSL handshake status code and the error string related to the code.

```
when CLIENTSSL_HANDSHAKE {
        log "SSL verify_result  CLIENTSSL_HANDSHAKE:status
code [SSL::verify_result] in the logs"
        log "SSL::verify result [X509::verify_cert_error_
string [SSL::verify_result]]"
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

SSLI::bypass

Description Use the following command to bypass SSL inspection.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
SSLI::bypass
```

Example Use the following example to bypass SSL inspection.

```
when SERVERSSL_SERVERCERT {
if { [SSL::cert issuer 1] contains "Digi" }  {
        log "SERVERSSL_SERVERCERT: bypass SSL"
        SSLI::bypass
    }
}
```

Valid Events

- SERVERSSL_SERVERCERT

SSLI::cache_cert

Description This command is used to disable or enable caching of server certificate.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
SSLI::cache_cert [disable|enable]
```

Example Use the following example to disable caching of server certificate.

```
when SERVERSSL_SERVERCERT {
    if { [SSL::cert issuer 1] contains "Digi" }  {
        log "SERVERSSL_SERVERCERT: SSLI::cache_cert disable"
        SSLI::cache_cert disable
    }
}
```

Valid Events

- SERVERSSL_SERVERCERT

SSLI::drop

Description Use the following command to drop the SSL connection.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
SSLI::drop
```

Example Use the following example to drop the SSL connection.

```
when SERVERSSL_SERVERCERT {
if { [SSL::cert issuer 1] contains "Digi" }  {
        log "SERVERSSL_SERVERCERT: drop SSL"
        SSLI::drop
    }
}
```

Valid Events

- SERVERSSL_SERVERCERT

SSLI::inspect

Description Use the following command to enable SSL inspection for the flow.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
SSLI::inspect
```

Inspects the flow.

```
SSLI::inspect use_alt_key
```

Inspects the flow and uses the alt key for signing.

Example Use the following example to inspect the flow and use the alt key for signing:

```
when SERVERSSL_SERVERCERT {
  if { [SSL::cert issuer 1] contains "a10-ssl.com" }  {
      log "SERVERSSL_SERVERCERT: inspect SSL"
      SSLI::inspect use_alt_key
   }
}
```

Valid Events

- SERVERSSL_SERVERCERT

Statistics Commands

The following commands related to statistics are supported:

- STATS::clear
- STATS::get

For information about aFleX commands, see aFleX Commands.

STATS::clear

Description Clears statistics for a real server (node), virtual server, or service group (pool).

Syntax

```
STATS::clear server <server-name | ipaddr> [<port-num> <tcp |
udp>]
current-connection | total-connection | request-pkt |
response-pkt
[partition shared]
```

Used to clear statistics for a real server.

```
STATS::clear virtual-server <vip-name| vipaddr> [<port-num>
<service-type>] current-connection | total-connection |
request-pkt |   response-pkt [partition shared]
```

Used to clear statistics for a virtual server.

```
STATS::clear pool <pool-name> [member <ipaddr> <port-num>]
current-connection | total-connection | request-pkt |
  response-pkt
[partition shared]
```

Used to clear statistics for a service group.

Example Use the following example to clear statistics.

```
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/clearstats" } {
        STATS::clear server rs1 80 tcp total-connection
        STATS::clear virtual-server vip1 80 http total-
connection
        STATS::clear pool example_service_group total-
connection
    }
}
```

Valid Events

All

For information about aFleX events, see aFleX Events.

STATS::get

Description Retrieves statistics for a real server (node), virtual server, or service group (pool).

Syntax

```
STATS::get server <server-name | ipaddr> [<port-num> <tcp |
udp>]
current-connection | total-connection | request-pkt |
response-pkt
[partition shared]
```

Used to retrieve statistics from a real server.

You can specify the server by its name or IP address (`<server-name>` or `<ipaddr>`). Optionally, you can specify an individual port by its port number (0-65535) and Layer 4 protocol (tcp or udp). By default, statistics for all the real ports of the server are returned.

To specify the types of statistics to return, use one of the following options:

- `current-connection`
- `total-connection`
- `request-pkt`
- `response-pkt`

The shared partition option applies the command to real servers in the shared partition. By default, the `STATS::get` command acts only upon the real servers located in the Role-Based Administration (RBA) partition that contains the aFleX policy.

```
STATS::get virtual-server <vip-name| vipaddr>
[<port-num> <service-type>]
current-connection | total-connection | request-pkt |
  response-pkt
[partition shared]
```

Used to retrieve statistics from a virtual server.

You can specify the virtual server by its name or VIP address (<vip-name> or <vipaddr>).

Optionally, you can specify an individual port by its port number (0-65535) and service type (tcp, udp, http, https, and so on). By default, statistics for all the ports of the virtual server are returned.

The other options are the same as those for real servers.

```
STATS::get pool <pool-name> [member <ipaddr> <port-num>]
current-connection | total-connection | request-pkt |
  response-pkt
[partition shared]
```

Used to retrieve statistics from a service group.

Specify the service group by its name (<pool-name>).

Optionally, you can specify an individual member (server and port) by the real server IP address and protocol port number. By default, statistics for all the members of the service group are returned.

The other options are the same as those for real servers and virtual servers.

Example

The following policy will select a real server based on the current connection counter:

```
when CLIENT_ACCEPTED {
        set total1 [STATS::get server 192.168.10.10 current-
connection]
        set total2 [STATS::get server 192.168.10.20 current-
connection]
        if { $total1 > $total2 } {
                node 192.168.10.20 80
        } else  {
                node 192.168.10.10 80
        }
}
```

Valid Events

All

Table Commands

You can use the following aFleX commands to manage a table of data entries:

- table add
- table append
- table delete
- table incr
- table keys
- table lifetime
- table lookup
- table replace
- table set
- table timeout

For an extended example using table commands, see Table Examples.

For information about aFleX commands, see aFleX Commands.

NOTE: The maximum number of elements allowed in a single table is 102,400 on any platform. This number is configurable by using the **system resource-usage aflex-table-entry-count** command in the CLI.

Table Entry Expiration Date

You can configure the `<lifetime>` and `<timeout>` values (from 1 to 4026531839 seconds) to predefine an expiration date for the table entries. The following list defines conditions for using these options.

Set an Indefinite Expiration Time – Use `indefinite` or `indef` for the `<timeout>` or `<lifetime>` parameters to allow an entry to remain in the table indefinitely. These entries will not expire and can only be removed from the table explicitly or when the ACOS device is rebooted.

Apply Existing Configuration – Specify the <lifetime> or <timeout> as 0 to use existing configuration. For new entries, this will set the <lifetime> or <timeout> to the default values.

Default Values

If <timeout> is not specified, the timeout is set to the default of 180 seconds.

If <lifetime> is not specified, the lifetime is set to “indefinite”.

NOTE: Table entries are not synced to peer units in HA configurations.

Depending on the aFleX event used in the policy, you can track connections or requests. The CLIENT_CONNECTED event represents TCP connections, whereas the HTTP_REQUEST event represents every individual request.

The <lifetime> option sets the entry to expire after the specified period of time, regardless of how many changes or lookups are performed on the entry.

An entry can have both a configured lifetime and timeout. The entry is removed from the table for whichever expiration time comes first.

table add

Description Adds or returns the value for a specified key in the table.

Syntax

```
table add <name> <key> <value> [<timout> [<lifetime>]]
```

Adds a key to the table with the specified `<key>` number and associated `<value>`. Optionally, you can apply a `<timeout>` and `<lifetime>` to the entry.

NOTE: If the key already exists, a key is not inserted and the existing value is not returned. When a new key is added, the existing value is returned.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table append

Description Appends a string to the value associated with the specified key

Syntax

```
table append <name> [-notouch] <key> <string>
```

If `-notouch` is specified, then any existing entries for the key will not have an updated timestamp.

NOTE: If the key does not exist, then no action is taken. This command returns the value of the entry after the operation is complete.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table delete

Description Deletes elements of a table.

Syntax

```
table delete <name> <key>
```

Deletes the `<key>` or value pair with the specified key.

```
table delete -all
```

Deletes all keys and value pairs for that table.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table incr

Description Increments the value associated with a key.

Syntax

```
table incr <name> [-notouch] <key> [<num>]
```

Increments the value associated with the `<key>`, in the specified table. If you do not specify a value for `<num>`, 1 is used by default. If `-notouch` is specified, then any existing entries for the key will not have an updated timestamp.

NOTE: This command returns the entry's value after the operation is complete. If the specified key does not exist, then no action is taken.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table keys

Description Returns a list of key and value pairs in the specified table.

Syntax

```
table keys <name>
```

Returns list of keys and value pairs for the table.

```
table keys <name> -count
```

Returns list of keys and value pairs (without updated timestamp), and number of keys in the specified table.

```
table keys <name> -notouch
```

Returns list of keys and value pairs without updated timestamp.

NOTE: A10 Networks does not recommend using this command frequently in an aFleX policy. The `table keys` command provides useful debugging capabilities, but can lower system performance when used repeatedly.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table lifetime

Description Returns the lifetime for the specified key. This command returns -1 if no lifetime is set for the specified key or the lifetime is indefinite.

Syntax

```
table lifetime <name> <key>
```

Returns the lifetime for `<key>`.

```
table lifetime <name> <key> <value>
```

Sets the lifetime for `<key>`.

```
table lifetime <name> -remaining <key>
```

Returns the remaining time before the expiration lifetime.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table lookup

Description Returns the value associated with the specified key.

Syntax

```
table lookup <name> <key>
```

Returns the value associated with `<key>`.

```
table lookup <name> -notouch <key>
```

Returns the value associated with `<key>`. Any existing entries for the key will not have an updated timestamp.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table replace

Description Replaces the value in the table associated with the specified key or value. If the specified key does not exist, no action is taken and an empty string is returned.

Syntax

```
table replace <name> <key> <value>
```

Replaces the value in the table with the specified `<key>` or `<value>`. Returns new value after replacement.

```
table replace <name> <key> <value> <timeout> <lifetime>
```

Replaces the value with the specified `<key>` or `<value>` and applies a `<timeout>` and `<lifetime>` to the entry. Returns new value after replacement.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table set

Description Sets a value in the table for an existing key. Adds a table and a key if one does not already exist.

Syntax

```
table set <name> <key> <value>
```

Sets the `<value>` of `<key>` and returns the entry's value.

```
table set <name> <key> <value> <timeout> <lifetime>
```

Sets the `<value>` of `<key>` and returns the entry's value. Also applies a `<timeout>` and `<lifetime>` to the entry.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

table timeout

Description Sets or returns the timeout for a specific key in a table.

Syntax

```
table timeout <name> <key>
```

Returns the timeout for `<key>`. Returns -1 if no timeout is set.

```
table timeout <name> <key> <value>
```

Sets the timeout for `<key>` to `<value>`.

```
table timeout <name> [-remaining] <key>
```

Returns the remaining time before timeout.

Example See Table Examples.

Valid Events

All.

For information about aFleX events, see aFleX Events.

Table Examples

The following aFleX script examples use table commands:

- Example 1 uses table commands to blacklist IP addresses that make large number of DNS queries.
- Example 2 uses table commands to block IP addresses if there are large number of failed login attempts.
- Example 3 shows an example of how to display and delete table commands.

Example 1

In this example, the aFleX script blacklists an IP addresses for 10 minutes (600 seconds) if traffic from the IP address makes more than 10 DNS queries per second. It uses the lifetime for the `$::HOLDTIME`:

```
when RULE_INIT {
    set ::MAXQUERY 10
    set ::HOLDTIME 600
}

when DNS_REQUEST {
    if { [table lookup "blacklist" [IP::client_addr]] != "" } {
        log "The Blacklist for [IP::client_addr] expires in [table
lifetime "blacklist" -remaining [IP::client_addr]] seconds"
        drop
        return
```

```
    }
    if { [table lookup tmp_table [IP::client_addr]] == "" } {
        table set tmp_table [IP::client_addr] 1 indef 1
        log "The table entry created for [IP::client_addr]"
        return
    }
    set count [table incr tmp_table [IP::client_addr]]
    log "The DNS Query $count of $::MAXQUERY for [IP::client_addr]"
    if { $count > $::MAXQUERY } {
        table add "blacklist" $key "blocked" indef $::HOLDTIME
        log "The Blacklist entry created for [IP::client_addr]"
        table delete tmp_table $key
        drop
        return
    }
}
```

Example 2

In this example, the aFleX script blocks an IP address for 10 minutes (600 seconds) if there are 3 failed login attempts. It uses the timeout for the `$::HOLDTIME`:

```
when RULE_INIT {
    set ::MAXTRIES 3
    set ::HOLDTIME 600
    set ::LOCATION "/welcome.cgi?p=failed"
}
when HTTP_REQUEST {
    if { [table lookup "failedlogins" -notouch [IP::client_addr]] ==
$::MAXTRIES } {
        HTTP::respond 200 content "You have been blocked, you can try
again in [table timeout "failedlogins" -remaining [IP::client_addr]]
seconds"
        log "Login is blocked for [IP::client_addr] expires in [table
timeout "failedlogins" -remaining [IP::client_addr]] seconds"
    }
}

when HTTP_RESPONSE {
    if { [HTTP::header exists "Location"] } {
        if { ([HTTP::header "Location"] ends_with $::LOCATION) } {
```

```
            if { [table lookup "failedlogins" [IP::client_addr]] != "" } {
                table incr failedlogins [IP::client_addr]
                table timeout "failedlogins" [IP::client_addr] $::HOLDTIME
            } else {
                table add failedlogins [IP::client_addr] 1 $::HOLDTIME
            }
            log "Login from [IP::client_addr] [table lookup "failedlogins"
-notouch [IP::client_addr]] of $::MAXTRIES remaining"
        }
    }
}
```

Example 3

In this example, the aFleX script presents all the entries in the table and gives the option to delete a table. Show table contents with: http://<vip>/status:<table-name>. Delete table contents with: http://<vip>/delete:<table-name>.

```
when HTTP_REQUEST {
    set ACTION [getfield [HTTP::uri] ":" 1]
    set TABLE [getfield [HTTP::uri] ":" 2]

    if { $ACTION eq "/flush" } {
        table delete $TABLE -all
        HTTP::respond 200 content "Table $TABLE deleted... <a
href=\"/status:$TABLE\">Back to STATUS</a>" Content-Type "text/html"
    } elseif { $ACTION eq "/status" } {
        set response "<html><head><title>Contents of Table:
$TABLE</title></head>"
        append response "<body><center><h1>Contents of Table:
$TABLE</h1><table border=\"1\" cellpadding=\"5\" cellspacing=\"0\">"
        append response "<tr><th>Key</th><th>Value</th></tr>"
        set i 0
        foreach tr [table keys $TABLE] {
            incr i
            if { $i == 1 } {
                append response "<tr><td>$tr</td>"
            }
            if { $i == 2 } {
                append response "<td>$tr</td></tr>"
                set i 0
```

```
            }
        }
        append response "</table><p>DELETE TABLE: <a
href=\"/flush:$TABLE\">$TABLE</a></p>"
        append response "</center></body></html>"
        HTTP::respond 200 content $response Content-Type "text/html"
    } else {
        HTTP::respond 200 content "Usage is prohibited!"
    }
}
```

TCP Commands

The following TCP commands are supported:

- TCP::client_port
- TCP::close
- TCP::collect
- TCP::local_port
- TCP::mss
- TCP::notify
- TCP::offset
- TCP::option
- TCP::payload
- TCP::release
- TCP::remote_port
- TCP::respond
- TCP::rtt
- TCP::server_port

For information about aFleX commands, see aFleX Commands.

For information about TCP events, see IP, TCP, and UDP Events.

TCP::client_port

Description This command will return the TCP port/service number of the specified client. It is equivalent to the command `clientside { TCP::remote_port }` and `client_port`.

Syntax

```
TCP::client_port
```

Example Use the following example to log the TCP port/service number of the specified client.

```
when CLIENT_ACCEPTED {
    log "Connection has been achieved here: [IP::client_addr]:
[TCP::client_port]"
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_AUTHORIZATION_INIT
- AAM_RELAY_INIT
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVERSSL_CLIENTHELLO_SEND

- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERCERT
- SERVERSSL_SERVERHELLO
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

TCP::close

Description This command will close the TCP connection.

NOTE: This command supports old SSL (N5) and new SSL (QAT, new N5, and Software TLS1.3).

Syntax

```
TCP::close
```

Example Use the following example to close the TCP connection.

```
when CLIENT_ACCEPTED {
    if { [IP::addr [IP::client_addr] equals 192.168.7.0/24] }
{
        TCP::close
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE

- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

TCP::collect

Description This command will cause TCP to start gathering the specified amount of content data.

For information about using this command with generic TCP Proxy, see Support for Generic TCP Proxy.

Syntax

```
TCP::collect <length>
```

The `<length>` parameter is used to specify the minimum number of bytes to collect.

Example Use the following example to collect minimum number of 15 bytes.

```
when CLIENT_ACCEPTED {
    TCP::collect 15
}
when CLIENT_DATA {
    if { [TCP::payload 15] contains "internal" } {
        pool service_group_internal
    } else {
        pool example_service_group
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_DATA
- SERVER_CONNECTED
- SERVER_DATA

Support for Generic TCP Proxy

aFleX also supports the generic tcp-proxy service type. This allows use of the `TCP::collect [<length>]` command to gather payload data on a tcp-proxy virtual port. If the `<length>` argument is not used, the script behaves a little differently.

TCP::collect <length>

When the `<length>` option is applied:

A DATA event will only be triggered if more than the specified number of collected data packets are available.

After a DATA event, `TCP::release` is implicitly forced to release data and will forward data packet, meaning a TCP release occurs even if the script has no `TCP::release` command.

Following the DATA event, the collect flag is disabled and TCP::collect will no longer be allowed in the DATA event.

In cases where the total TCP payload is less than the length specified, the ACOS device will continue to wait for more data from the client, and its expected action to forward the information to the server will not occur. It is important to specify the correct length value.

Example

```
when CLIENT_ACCEPTED {
    TCP::collect 1024
}
when CLIENT_DATA {
    log "Here is the length of the payload: [TCP::payload
length]"
    if { [TCP::payload 15] contains "internal" } {
        pool service_group_internal
    } else {
        pool example_service_group
    }
}
```

TCP::collect

When the `<length>` option is not specified:

Upon receipt of the first data packet, a DATA event is triggered.

The ACOS device buffers data instead of forwarding it when an event does not contain the command `TCP::release`.

Another `TCP::collect` command must follow the `TCP::release` command to have a script continue to gather the next packet.

The collect flag is disabled when the DATA event does not have a `TCP::collect` command, so no incoming packets will be gathered.

Example

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    log "Here is the length of the payload: [TCP::payload
length]"
    if { [TCP::payload 15] contains "internal" } {
        pool service_group_internal
    } else {
        pool example_service_group
    }
    TCP::release
    TCP::collect
}
```

Server Selection Behavior if TCP::collect [<length>] Command Is Not Used with Generic TCP-Proxy Traffic

When the `TCP::collect [<length>]` command is not used, the ACOS devices will do server selection after establishment of a client session and send SYN to the server for generic TCP-proxy traffic.

Th client will send SYN.

The ACOS device will send SYN-ACK.

The client will send ACK.

Upon receiving the ACK from the client, the ACOS device will select a real server and send a SYN to the server.

The session flow will continue with the selected server.

When `TCP::collect` is used in an `CLIENT_ACCEPTED` event, the ACOS device will not be able to start a connection with a back-end server after the client ACK.

In this situation, the aFleX collect operation must be completed and the `CLIENT_DATA` event must be triggered. Before implementing these actions, ensure that the ACOS device is on standby. After the collection of data is done, the ACOS device will select a real server and forward its data.

The client will send SYN.

The ACOS device will send SYN-ACK.

The client will send ACK.

The client data is pushed.

When the collect operation is complete, trigger `CLIENT_DATA` event will occur and a connection will be established to a selected server.

When the collect operation is not complete, client data will continue to be buffered and will finally be forwarded to a server when collect operation is completed. (completion defined by collect <length>)

Additional Generic TCP-Proxy Examples

Example This example illustrates the collection of just the first data packet and trigger of DATA event.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    log "Here is the length of payload: [TCP::payload length]"
    if { [TCP::payload ] contains "internal" } {
        pool service_group_internal
    } else {
        pool example_service_group
    }
    TCP::release
}
```

Example Use the following example to set and log the TCP payload length. Check if the payload contains a certain string and use the appropriate service group for the content. Alternatively, use a different service group and release it. This example shows the gathering of the first 1000 bytes of data and triggering a DATA event using the following key commands:

- `TCP::collect` command using `<length>` option in `CLIENT_ACCEPTED` event
- `TCP::release` command placed at the end of `CLIENT_DATA` event

```
when CLIENT_ACCEPTED {
        TCP::collect 1000
}
when CLIENT_DATA {
        set tcplen [TCP::payload length]
        log "Here is the length : ($tcplen)"
        if { [TCP::payload ] contains "ABC" } {
                pool abc_service_group
        } else {
                pool web_service_group
        }
        TCP::release
}
```

NOTE:

- Ensure the correct `<length>` value, otherwise if the TCP payload total is less than that specified by collect <length>, the ACOS device does not forward the data to the server.
- aFleX does not allow another use of the `TCP::collect` command in the DAT event when collect `<length>` is defined.

Example This example shows the gathering of the first three data packets followed by the action of forwarding the data to the server by using these commands:

- `TCP::collect` command used in `CLIENT_ACCEPTED` event

- In the `CLIENT_DATA` event, `TCP::release` executes when the number of gathered packets is greater than three.

```
when CLIENT_ACCEPTED {
    TCP::collect
    set packet_count 0
}
when CLIENT_DATA {
    incr packet_count
    if { $packet_count >= 3 }  {
        log "Here is the length of the payload: [TCP::payload
length]"
        if { [TCP::payload] contains "internal" } {
            pool service_group_internal
        } else {
            pool example_service_group
        }
    TCP::release
    }
}
```

Valid Events

The following events are valid for this use of the `TCP::collect` command:

- CLIENT_ACCEPTED
- CLIENT_DATA
- SERVER_CONNECTED
- SERVER_DATA

TCP::local_port

Description This command will return the local TCP port/service number.

Syntax

```
TCP::local_port
```

Example

This example shows a cookie modification using the local port information. If the port is 80, the cookie has "HttpOnly" added to it. If the port is 443, "HttpOnly" and "Secure" is added to the cookie.

```
when RULE_INIT {
  set ::DEBUG 0
}

when HTTP_REQUEST {
  set PORT [TCP::local_port]
}

when HTTP_RESPONSE {
  set current_time [TIME::clock seconds]
  foreach cookie_name [HTTP::cookie names] {
    if { [HTTP::cookie exists "$cookie_name"] } {
      set new_cookie "$cookie_name=[HTTP::cookie value
"$cookie_name"]"
      if { [HTTP::cookie expires "$cookie_name"] > $current_
time } {
        set cookie_expires [clock format [HTTP::cookie expires
"$cookie_name"] -format {%a, %d %b %Y %H:%M:%S GMT} -gmt 1]
        append new_cookie "; Expires=$cookie_expires" }
      if { [HTTP::cookie domain "$cookie_name"] ne "" } {
append new_cookie "; Domain=[HTTP::cookie domain "$cookie_
name"]" }
      if { [HTTP::cookie path "$cookie_name"] ne "" } { append
new_cookie "; Path=[HTTP::cookie path "$cookie_name"]" }
      if { $PORT == 443 } { append new_cookie "; Secure" }
      if { $PORT == 80 or $PORT == 443 } { append new_cookie
"; HttpOnly" }
      if { ($::DEBUG == 1) } { log "Set-Cookie $new_cookie" }
      HTTP::cookie remove "$cookie_name"
      HTTP::header insert Set-Cookie "$new_cookie"
    }
  }
}
```

Valid Events

- AAM_AUTHENTICATION_INIT

- AAM_AUTHORIZATION_CHECK
- AAM_AUTHORIZATION_INIT
- AAM_RELAY_INIT
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERCERT
- SERVERSSL_SERVERHELLO
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

TCP::mss

Description This command will return the maximum segment size (MSS) for a TCP connection.

Syntax

```
TCP::mss
```

Example Use the following example to log the MSS for a TCP connection.

```
when CLIENT_ACCEPTED {
  log "Here is the maximum segment size: [TCP::mss]"
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_AUTHORIZATION_INIT
- AAM_RELAY_INIT
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED

- SERVER_DATA
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERCERT
- SERVERSSL_SERVERHELLO
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

TCP::notify

Description This command will notify the system that the end of a message has been reached, and that the message is ready for load balancing.

Syntax

```
TCP::notify eom
```

Example This example shows how to use notify for load balancing messages through TCP.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    log "Here is the payload: [TCP::payload] "
    TCP::release
    TCP::notify eom
    log "Here is the payload after release: [TCP::payload]"
    TCP::collect
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

TCP::offset

Description This command will return the position in the TCP data stream where the collected TCP data began.

Syntax

```
TCP::offset
```

Example Use the following example to return the position in the TCP data stream.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    if { [TCP::offset] > 1000 } {
        TCP::release
    }
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

TCP::option

Description This command will retrieve, set, or unset the raw value of the specified option kind from the TCP header.

Some common option kinds are listed:

- Option kind 2 - Max Segment Size (MSS)
- Option kind 3 - Window Scale
- Option kind 4 - SACK Permitted (Selective Acknowledgment)
- Option kind 5 - SACK
- Option kind 8 - Timestamps

Syntax

```
TCP::option [get <option> | set <option> <value> | unset
<option> ]
```

The <option> parameter is a numeric value that indicates the TCP option kind. The different options used are described below:

Kind	Length	Meaning
0	-	End of Option List
1	-	No-Operation
2	4	Maximum Segment Size
3	3	Window Scale
4	2	SACK Permitted
5	N	SACK
6	6	Echo (obsoleted by option 8)
7	6	Echo Reply (obsoleted by option 8)
8	10	Timestamps
9	2	Partial Order Connection Permitted (obsolete)
10	3	Partial Order Service Profile (obsolete)
11		CC (obsolete)
12		CC.NEW (obsolete)
13		CC.ECHO (obsolete)
14	3	TCP Alternate Checksum Request (obsolete)
15	N	TCP Alternate Checksum Data (obsolete)
16		Skeeter
17		Bubba
18	3	Trailer Checksum Option
19	18	MD5 Signature Option (obsoleted by option 29)
20		SCPS Capabilities
21		Selective Negative Acknowledgements
22		Record Boundaries
23		Corruption experienced
24		SNAP

Kind	Length	Meaning
25		Unassigned (released 2000-12-18)
26		TCP Compression Filter
27	8	Quick-Start Response
28	4	User Timeout Option (also, other known unauthorized use) [***][1]
29		TCP Authentication Option (TCP-AO)
30	N	Multipath TCP (MPTCP)
31		Reserved (known unauthorized use without proper IANA assignment) [**]
32		Reserved (known unauthorized use without proper IANA assignment) [**]
33		Reserved (known unauthorized use without proper IANA assignment) [**]
34	vary	TCP Fast Open Cookie
35-75		Reserved
69		Reserved (known unauthorized use without proper IANA assignment) [**]
70		Reserved (known unauthorized use without proper IANA assignment) [**]
71-75		Reserved
76		Reserved (known unauthorized use without proper IANA assignment) [**]
77		Reserved (known unauthorized use without proper IANA assignment) [**]
78		Reserved (known unauthorized use without proper IANA assignment) [**]
79-252		Reserved

Kind	Length	Meaning
253	N	"RFC3692-style Experiment 1 (also improperly used for shipping products) [*]"
254	N	"RFC3692-style Experiment 2 (also improperly used for shipping products) [*]"

NOTE: For the following TCP option types, they cannot be set or unset: MSS, SACK and Window Scale.

Example Use this example to get the value for different options set by client in the TCP header.

```
when CLIENT_ACCEPTED {
    log " TS = [TCP::option get 8]"
    log " mss = [TCP::option get 2]"
    log " wscale = [TCP::option get 3]"
    log " SACK_permit = [TCP::option get 4]"
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

TCP::payload

Description This command will return the accumulated TCP data content, or replace the gathered payload with the specified data.

Syntax

```
TCP::payload [<size>]
```

This will return the accumulated TCP data content.

```
TCP::payload <offset> <size>
```

This will return the accumulated TCP data content start from <offset>.

```
TCP::payload length
```

This will return the amount of accumulated TCP data content in bytes.

```
TCP::payload <offset> <size> <data>
```

This will return the gathered payload with the specified data.

```
TCP::payload replace <offset> <size> <data>
```

This will replace the gathered payload with the specified data.

NOTE: Use of the `TCP::payload replace` command is only supported for TCP-proxy, TCP, and FTP virtual ports.

NOTE: After TCP data has been released with the use of the `TCP::release` command, it is no longer part of the TCP data payload, so it will not be returned with the `TCP::payload` command.

Example Use the following example to return the accumulated TCP data content.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    if { [TCP::payload] contains "internal" } {
        pool service_group_internal
    } else {
        pool service_group_tcp
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA

- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

TCP::release

Description This command will cause TCP to resume processing the connection and flush collected data.

Syntax

```
TCP::release
```

This will release the collected TCP payload.

```
TCP::release <size>
```

This will release the specified amount of the TCP payload.

NOTE: Use `TCP::release <size>` only with MLB-TCP virtual ports.

NOTE: After TCP data is released using the `TCP::release` command, it is no longer part of the TCP data payload, so it will not be returned with the `TCP::payload` command.

Example Use the following example to resume processing the connection and flush collected data.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
when CLIENT_DATA {
    if { [TCP::offset] > 1000 } {
        TCP::release
    }
}
```

Example Use the following example to show message load balancing to help determine the proper payload.

```
when CLIENT_ACCEPTED {
    TCP::collect
}
```

```
when CLIENT_DATA {
    log "Here is the payload: [TCP::payload] "
    TCP::release 20
    TCP::notify eom
    log "Here is the payload after release: [TCP::payload]"
    TCP::collect
}
```

Valid Events

- CLIENT_DATA
- SERVER_DATA

TCP::remote_port

Description This command will return the remote TCP port/service number. It replaces the `remote_port` command.

If used with the clientside command, specifically `clientside TCP::remote_port`, the `TCP::remote_port` command is the same as using the TCP::client_port command.

If used with the serverside command, specifically `serverside TCP::remote_port`, the `TCP::remote_port` command is the same as using the TCP::server_port command.

Syntax

```
TCP::remote_port
```

Example Use the following example to log the remote TCP port.

```
when SERVER_CONNECTED {
    log "Here is the server remote TCP port: [TCP::remote_
port]"
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED

- SERVER_CONNECTED
- SERVER_DATA

TCP::respond

Description This command will send the specified data directly to the peer. It can also be used to complete a protocol handshake.

NOTE:

- This command supports only old SSL (N5) and does not support new SSL (QAT, new N5, and Software TLS1.3).
- This command will not work if applied to an HTTP/HTTPS virtual port.

Syntax

```
TCP::respond <data>
```

This command with the `<data>` parameter will specify the data to send to the peer.

NOTE: This command will not work if applied to an HTTP virtual port.

Example Use the following example to locate an “EHLO <hostname>” command and have it respond with a specific error message. This aFleX script will intercept the TCP stream between a back-end server and an SMTP server.

```
when CLIENT_ACCEPTED {
    TCP::collect 4
}
when CLIENT_DATA {
    if { [TCP::payload] starts_with "EHLO" } {
        TCP::respond "500 5.3.3 Unrecognized command\r\n"
    }
    TCP::release
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED

- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

TCP::rtt

Description This command will return the smoothed round-trip time (RTT) estimate for a TCP connection.

Syntax

```
TCP::rtt
```

- Get the actual round-trip time in milliseconds by dividing the returned value by 2.
- The RTT will take some time to converge.

Example Use the following example to get the actual round-trip time in milliseconds.

```
when HTTP_REQUEST {
    set rtt [TCP::rtt]
}
when HTTP_RESPONSE {
    if { $rtt < 1600 } {
        log "Here is the round-trip time: $rtt for
[IP::client_addr] - without compress applied."
        COMPRESS::disable
    } else {
        log "Here is the round-trip time: $rtt for
[IP::client_addr] - with compress applied."
        COMPRESS::enable
        COMPRESS::gzip level 3
    }
}
```

Valid Events

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_AUTHORIZATION_INIT

- AAM_RELAY_INIT
- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- CLIENTSSL_CLIENTCERT
- CLIENTSSL_CLIENTHELLO
- CLIENTSSL_DATA
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_FAILED
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SERVERSSL_CLIENTHELLO_SEND
- SERVERSSL_DATA
- SERVERSSL_HANDSHAKE
- SERVERSSL_SERVERCERT
- SERVERSSL_SERVERHELLO
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

TCP::server_port

Description This command will return the TCP port/service number of the specified server. It is the same as using the `serverside { TCP::remote_port }` command and the obsolete variable server_port.

Syntax

```
TCP::server_port
```

Example Use the following example to log the TCP port of the specified server.

```
when SERVER_CONNECTED {
    log "Here is the server port: [TCP::server_port]"
}
```

Valid Events

- AAM_RELAY_INIT
- CLIENT_CLOSED
- CLIENT_DATA
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA
- LB_SELECTED
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA
- SIP_REQUEST
- SIP_REQUEST_SEND
- SIP_RESPONSE

Template Commands

The Template commands enable you to access individual configuration parameters on a per template basis. The commands listed below allow you to check for the existence of certain template types on a virtual server. Further, you can use these commands to access configuration parameters for a template.

The following template commands are supported:

- TEMPLATE::cache
- TEMPLATE::client_ssl
- TEMPLATE::conn_reuse
- TEMPLATE::exists
- TEMPLATE::http
- TEMPLATE::server_ssl
- TEMPLATE::tcp
- TEMPLATE::udp

For information about aFleX commands, see aFleX Commands.

TEMPLATE::cache

Description This command gets the value of the parameter for a RAM cache template.

Syntax

```
TEMPLATE::cache <setting>
```

This command returns the value for the specified `<setting>` in the designated RAM cache template. For `the <setting>` variable, you can enter one of the following options:

- `name`
- `accept_reload_req`
- `age`
- `default_policy_nocache`
- `disable_insert_age`
- `disable_insert_via`
- `max_cache_size`
- `max_content_size`
- `min_content_size`
- `policy`
- `remove_cookies`
- `replacement_policy`
- `verify_host`

Example The following example logs the cache age value in the assigned template.

```
when CACHE_REQUEST {
    log "Cache age of URI [HTTP::uri] refreshed to
[TEMPLATE::cache age]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::client_ssl

Description This command gets the value of the parameter for the client SSL template.

Syntax

```
TEMPLATE::client_ssl <setting>
```

This command returns the value for the specified `<setting>` in the designated client SSL template. For the `<setting>` parameter, enter one of the following options:

- `name`
- `ca_cert`
- `cert`
- `chain_cert`
- `cipher`
- `client_certificate`
- `close_notify`
- `crl`
- `key`
- `session_cache_size`
- `ssl_false_start_disable`

Example The following example checks if a client-side SSL template exists on the virtual port, and if the SSL template is found, then a log is generated containing the template name.

```
when CLIENT_ACCEPTED {
    if { [TEMPLATE::exists client_ssl] == 1 } {
        log "The TEMPLATE exists and says YES."
        log "The TEMPLATE Client SSL name is
[TEMPLATE::client_ssl name]."
    } else {
        log "No, the client SSL Template is not configured."
    }
```

```
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::conn_reuse

Description This command gets the value of the parameter for the connection reuse template.

Syntax

```
TEMPLATE::conn_reuse <setting>
```

This command returns values for the specified `<setting>` in the designated connection reuse template. For the `<setting>` variable, you may enter one of the options below:

- `name`
- `keep_alive_conn`
- `limit_per_server`
- `timeout`

Example The following example checks if a connection reuse template exists and if the current number of connections is greater than the limit per server. If so, it forwards the traffic to a special service port.

```
when HTTP_REQUEST {
    if { [TEMPLATE::exists conn_reuse] == 1 } {
        set curr_conn [STATS::get server 192.168.1.1 80 tcp
current-connection]
        if { $curr_conn > [TEMPLATE::conn_reuse limit-per-
server] } {
            node 192.168.1.100 80
        }
    }
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::exists

Description This command determines whether a template is bound to a virtual server. The command returns a “1” integer value if a template is configured on the virtual server, and it returns a “0” integer value if the template is not configured on the virtual server.

Syntax

```
TEMPLATE::exists [cache | client_ssl | conn_reuse | http |
server_ssl | tcp | udp]
```

This command checks for the presence of the templates as mentioned below:

- `cache` – RAM Caching template
- `client_ssl` – Client SSL template
- `conn_reuse` – Connection Reuse template
- `http` – HTTP template
- `server_ssl` – Server SSL template
- `tcp` – TCP template or TCP proxy template
- `udp` – UDP template

```
TEMPLATE::exists persist [cookie | src_ip | dst_ip | ssl_sid]
```

The command checks if the following types of persistence templates exist:

- `cookie` – Cookie Persistence
- `src_ip` – Source IP Persistence
- `dst_ip` – Destination IP Persistence
- `ssl_sid` – SSL Session ID Persistence

Example The following example checks if a Client SSL template has been applied to the virtual server. If yes, then the command will return a “1” value, and this will trigger ACOS to create a log message indicating that the client SSL template is enabled.

```
when CLIENT_ACCEPTED {
```

```
    if { [TEMPLATE::exists client_ssl] == 1 } {
        log "The client SSL template is configured."
    }
}
```

Example The following example checks if a Server SSL template has been applied to the virtual server. If yes, then the command will return a “1” value, and this will trigger ACOS to create a log message indicating that the server SSL template is enabled.

```
when SERVER_CONNECTED {
    if { [TEMPLATE::exists server_ssl] == 1 } {
        log "The server SSL template is configured."
    }
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::http

Description This command gets the value of the parameter for the HTTP template.

Syntax

```
TEMPLATE::http <setting>
```

This command returns the value for the specified `<setting>` for the designated HTTP template. For the `<setting>` parameter, enter one of the options listed below:

- `name`
- `compress_level`
- `compress_content_type_excludes`
- `compress_uri_excludes`
- `compress_enable`
- `compress_min_size`
- `compress_content_type`
- `failover_url`

- host_switching
- insert_client_ip
- log_retry
- redirect_rewrite
- request_header_erase
- request_header_insert
- response_header_erase
- response_header_insert
- retry_on_5xx
- retry_on_5xx_per_req
- strict_transaction_switch
- term_11client_hdr_client_close
- url_hash_persist
- url_switching

Example

The following example checks if an HTTP template exists. If so, it skips the compression for the URI that contains the string “example” when the compression level is 1.

```
when HTTP_REQUEST {
    if { [TEMPLATE::exists http] == 1 } {
        #skip low level compression for certain uri
        if { [HTTP::uri] contains "example" } {
            if { [TEMPLATE::http comparess_level] == 1 } {
                COMPRESS::disable
            }
        }
    }
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::persist

Description This command enables the persistence of the configuration or data associated with the template. It allows the template's configuration to be stored across sessions or across multiple requests.

Syntax

```
TEMPLATE::persist <template-name> <true/false>
```

Example Use the following example to enable and disable template persistence based on client connection status:

```
when CLIENT_ACCEPTED {
    TEMPLATE::persist "my-template" true
    log "Template persistence is enabled for my-template."
}

when CLIENT_DISCONNECTED {
    TEMPLATE::persist "my-template" false
    log "Template persistence is disabled for my-template."
}
```

TEMPLATE::port

Description This command allows to specify a particular port for the template. It allows you to configure which port should be used when applying a template for the given traffic.

Syntax

```
TEMPLATE::port <port-number>
```

Example Use the following example to apply a template to traffic on port 80 (HTTP):

```
when CLIENT_ACCEPTED {
    TEMPLATE::port 80
    log "Template applied to port 80 for HTTP traffic"
}
```

TEMPLATE::server

Description This command defines or modifies the server template for a specific server instance.

Syntax

```
TEMPLATE::server <server-name>
```

Example Use the following example to apply a server template for a specific server named `server1`:

```
when HTTP_REQUEST {
    TEMPLATE::server "server1"
}
```

TEMPLATE::server_ssl

Description This command gets the value of the parameter for the server SSL template.

Syntax

```
TEMPLATE::server_ssl <setting>
```

This command returns the value for the specified `<setting>` for the designated Server SSL template. For the `<setting>` parameter, enter one of the options listed below:

- `name`
- `ca_cert`
- `cert`
- `cipher`
- `close_notify`
- `key`
- `version`

Example The following example checks if a Server SSL template exists, then logs all the listed settings, otherwise logs that it is not configured on the virtual port.

```
when CLIENT_ACCEPTED {
        if { [TEMPLATE::exists server_ssl] == 1 } {
```

```
                log "*** Template server_ssl is configured on vport ***"
                log "*** Name: [TEMPLATE::server_ssl name]***"
                log "*** ca_cert: [TEMPLATE::server_ssl ca_cert]***"
                log "*** cert: [TEMPLATE::server_ssl cert]***"
                log "*** close_notify: [TEMPLATE::server_ssl close_
notify]***"
                log "*** cipher: [TEMPLATE::server_ssl cipher]***"
                log "*** version: [TEMPLATE::server_ssl version]***"
                log "*** key: [TEMPLATE::server_ssl key]***"
        } else {
        log "template server_ssl is not configured on vport"
        }
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::tcp

Description This command gets the value of the parameter for the TCP template.

Syntax

```
TEMPLATE::tcp <setting>
```

This command returns the value for the specified `<setting>` in the designated TCP template. For `the <setting>` parameter, enter one of the choices listed below:

- `name`
- `force_delete_timeout`
- `half_close_idle_timeout`
- `idle_timeout`
- `initial_window_size`
- `reset_fwd`
- `reset_rev`

Example The following example logs the TCP idle timeout.

```
when CLIENT_ACCEPTED {
    log "Idle time out: [TEMPLATE::tcp idle_timeout]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

TEMPLATE::tcp proxy

Description This command sets up a TCP proxy to direct incoming TCP traffic through a specified template.

Syntax

```
TEMPLATE::tcp_proxy <template_name>
```

Example Use the following example to configure a TCP proxy using a specific template for handling TCP traffic:

```
when CLIENT_ACCEPTED {
    TEMPLATE::tcp_proxy "tcp_template_1"
}
```

TEMPLATE::udp

Description This command gets the value of the parameter for the UDP template.

Syntax

```
TEMPLATE::udp <setting>
```

This command returns the value for the specified `<setting>` for the designated UDP template. For the `<setting>` parameter, enter one of the options below:

- `name`
- `aging`
- `idle_timeout`
- `qos`
- `re_select_if_server_down`
- `stateless_conn_timeout`

Example The following example logs the UDP idle timeout.

```
when CLIENT_ACCEPTED {
    log "Idle time out: [TEMPLATE::udp idle_timeout]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

Time Commands

The time commands are used to return and format the time.

The following time commands are supported:

- TIME::clock

For information about aFleX commands, see aFleX Commands.

TIME::clock

Description This command returns the system time in seconds or milliseconds.

Syntax

```
TIME::clock [seconds | milliseconds | format | scan]
```

This command is recommended for use in SMP environments to facilitate high-performance processing. The lowest resolution of the timer is 4 milliseconds.

```
TIME::clock seconds
```

Returns the current system time in seconds

```
TIME::clock milliseconds
```

Returns the current system time in milliseconds

```
TIME::clock format <time> -format [<descriptors>] [-gmt <True | False>]
```

Converts a numeric time value (from TIME::clock) into a human-readable formatted string. Use `-gmt True` to return time in GMT. If `-gmt` is omitted or set to `False`, local time is returned. If no descriptors are specified after `-format`, the default format returned is:
`"%a %b %d %H:%M:%S %Z %Y."`
See the following table for details about field descriptors.

```
TIME::clock scan <datestring> [-base <clockvalue>] [-gmt <True | False>]
```

Parses a date/time string and converts it into a numeric clock value (epoch time).

Descriptor	Returns
%%	Inserts a “%.”
%a	Weekday, abbreviated (Mon, Tues, Wed, etc.).
%A	Weekday, unabbreviated (Monday, Tuesday, etc.).
%b	Month, abbreviated (Jan, Feb, etc.).
%B	Month, unabbreviated (January, February, etc.).

Descriptor	Returns
%c	Locale specific date and time.
%C	First two digits of the year (19, 20, etc).
%d	Day of the month, with leading zero if necessary (01 - 31).
%D	Date, in format “%m/%d/%y.”
%e	Day of month, without leading zeros (1 - 31).
%g	The ISO year (corresponding to the ISO week, “%V”), expressed as a two-digit year-of-the-century, with leading zero if necessary.
%G	The ISO year corresponding to the ISO week (%V), expressed as a four-digit number.
%h	Month name, abbreviated (Jan, Feb, etc.).
%H	Hour, 24-hour format, with leading zeros if necessary (00-23).
%I	Hour, 12-hour format, with leading zeros if necessary (01-12).
%j	Day of the year, with leading zeros if necessary (000-366).
%k	Hour, 24-hour format, no leading zeros (0-23).
%l	Hour, 12-hour format, no leading zeros (1-12).
%m	Month, as a number (01-12).
%M	Minute (00-59).
%n	Line break.
%p	Displays AM or PM.
%r	Time in a locale-specific "meridian" format. The "meridian" format in the default "C" locale is "%I:%M:%S %p".
%R	Time in hours and minutes (same as “%H:%M).
%s	Number of seconds since the `TIME::clock` command was executed.

Descriptor	Returns
%S	Seconds (00-59).
%t	Tab.
%T	Displays time in hours, minutes and seconds (same as “%H:%M:%S”).
%u	Weekday, as a number (Monday=1, Sunday=7).
%U	Week of the year, with Sunday as first day of the week (00-52).
%V	Week of the year according to ISO rules (The week including January 4 is week 1).
%w	Weekday, as a number (Sunday=0, Saturday=6).
%W	Week of the year, with Monday as first day of the week (00-52).
%x	Locale specific date format.
%X	Locale specific 24-hour time format.
%y	Last two digits of the year (00-99).
%Y	Four-digit year (for example, 1985)
%Z	Time zone.

Example Use this example to log the current system time when a TCP connection is established:

```
when CLIENT_ACCEPTED {
    set current_time [TIME::clock]
    log "Current system time (epoch): $current_time"
}
```

Example Use this example to log the time a client connection is established, formatted as a 4-digit year, month, day, and time (HH:MM:SS):

```
when CLIENT_ACCEPTED {
    log "The client [IP::client_addr] connected at
[TIME::clock format [TIME::clock seconds] -format {%Y/%m/%d at
%H:%M:%S}]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

UDP Commands

The following link commands are supported:

- UDP::client_port
- UDP::local_port
- UDP::payload
- UDP::remote_port
- UDP::respond
- UDP::server_port

For information about aFleX commands, see aFleX Commands.

For information about UDP events, see IP, TCP, and UDP Events.

UDP::client_port

Description This command returns the UDP port/service number for the designated client. It is equivalent to the command `clientside { UDP::remote_port }`.

Syntax

```
UDP::client_port
```

Example Use the following example when a client has established a connection if the client port is UDP 123, then use the service group for UDP.

```
when CLIENT_ACCEPTED {
    if { [UDP::client_port] == 123 } {
        pool service_group_udp
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

UDP::local_port

Description This command returns the local UDP port/service number.

Syntax

```
UDP::local_port
```

Example Use the following example when a client has established a connection to use service group for DNS if the local UDP port is 53. Otherwise, if the local port is UDP 123, then use the service group for UDP. If anything else, then drop.

```
when CLIENT_ACCEPTED {
    if { [UDP::local_port] == 53 } {
```

```
        pool service_group_dns
    } elseif { [UDP::local_port] == 123 } {
        pool service_group_udp
    } else {
        drop
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

UDP::mss

Description This command returns or modifies the Maximum Segment Size (MSS) for UDP packets.

Syntax

```
UDP::mss
```

Example Use the following example to log the MSS for a received UDP packet:

```
when UDP_REQUEST {
    set mss [UDP::mss]
    log "Maximum Segment Size for UDP packet: $mss"
}
```

UDP::payload

Description This command returns the content or length of the current UDP payload.

Syntax

```
UDP::payload [<size>]
```

This option returns the content of the current UDP payload.

```
UDP::payload length
```

This option returns the length, in bytes, of the current UDP payload.

```
UDP::payload <offset> <size>
```

This option returns the content of the current UDP payload from <offset>.

```
UDP::payload replace <offset> <size> <new_data>
```

Starting at <offset>, the option replaces the <size> of the collected payload with the specified <new_data>.

Example Use the following example to use dns_service_group1 when the UDP payload index from 12 through 20 contains the example string, else use dns_service_group2.

```
when CLIENT_DATA {
        if { [UDP::payload 12 20] contains "example string" } {
                pool dns_service_group1
        } else {
                pool dns_service_group2
        }
}
```

Example In the following example, the payload is emptied and then re-filled with the data from the “packetdata” string that is sent to the server.

```
when CLIENT_DATA {
        UDP::payload replace 0 [UDP::payload length] ""
        # craft a string to hold data, 0x01 0x00 0x00 0x00 0x02 0x00
0x00 0x00 0x03 0x00 0x00 0x00
        set packetdata [binary format i1i1i1 1 2 3 ]
        UDP::payload replace 0 0 $packetdata
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED

- SERVER_CONNECTED
- SERVER_DATA

UDP::remote_port

Description This command returns the remote UDP port/service number.

Syntax

```
UDP::remote_port
```

Example Use the following example to use service_group_udp if the UDP remote port equals 123.

```
when CLIENT_ACCEPTED {
    if { [UDP::remote_port] == 123 } {
        pool service_group_udp
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

UDP::respond

Description This command sends the specified data directly to the peer. You can use this command to complete the protocol handshake.

Syntax

```
UDP::respond <data>
```

The `<data>` parameter specifies the data to be sent to the peer.

Example Use the following example if the UDP payload contains one string and you want to respond with another string.

```
when CLIENT_DATA {
```

```
        if { [UDP::payload] contains "asd"] } {
                UDP::respond "jkl"
        }
}
```

Example Use the following example to compare the client address to the network address, then drop if it matches, and send an error message to the peer.

```
when CLIENT_DATA {
    if { [IP::addr [IP::client_addr] equals 192.168.0.0] } {
        UDP::drop
        UDP::respond "Error: The client is not allowed\r\n"
    }
}
```

Example Use the following example to initialize the data with the required binary format and respond to the peer with it.

```
when CLIENT_ACCEPTED {
        set packet [binary format S {0x0000}]
        UDP::respond $packet
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

UDP::server_port

Description This command returns the UDP port/service number of the server. This command is equivalent to the command `serverside { UDP::remote_port }`.

Syntax

```
UDP::server_port
```

Example Use the following example if the UDP server port equals 123, then log it.

```
when SERVER_CONNECTED {
    if { [UDP::server_port] == 123 } {
        log "The Server Port is [UDP::server_port]."
    }
}
```

Valid Events

- CLIENT_ACCEPTED
- CLIENT_CLOSED
- CLIENT_DATA
- SERVER_CLOSED
- SERVER_CONNECTED
- SERVER_DATA

URI Commands

The following commands can be used to return URI information:

- URI::basename
- URI::decode
- URI::encode
- URI::params
- URI::path
- URI::query

For information about aFleX commands, see aFleX Commands.

URI::basename

Description This command returns the basename portion for the designated URI. For example, given the URI `/path/to/file.ext?=param=value`, `URI::basename` returns `file.ext`

Syntax

```
URI::basename <uri>
```

Example Use the following example to log the basename portion of the URi for an HTTP request.

```
when HTTP_REQUEST {
    log "The URI Basename is [URI::basename [HTTP::uri]]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

URI::decode

Description This command returns a decoded version for a specified URI.

Syntax

```
URI::decode <uri>
```

Example Decodes a known URL encoded string and logs the output.

```
when HTTP_REQUEST {
  set d "wtf%20%30%31%32"
  set e [URI::decode $d]
  log "uri decode HTTP_REQUEST:$e"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA

- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

URI::encode

Description This command returns an encoded version of a designated URI.

Syntax

```
URI::encode <uri>
```

Example Use the following example to encode the URIs in HTTP requests and/or responses such that the URI associated with the 404 redirect message is encoded.

```
when HTTP_REQUEST {
    set HOST [HTTP::host]
}
when HTTP_RESPONSE {
    if { [HTTP::status] == 404 } {
        HTTP::redirect http://backup.example.com/?q=
[URI::encode "redirected by $HOST"]
    }
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

URI::params

Description This command returns the parameters from the URI (Uniform Resource Identifier) of the current HTTP request.

Syntax

```
URI::params
```

Example Use the following example to log the parameters from the URI when an HTTP request is received:

```
when HTTP_REQUEST {
    set params [URI::params]
    log "Request URI parameters: $params"
}
```

URI::path

Description This command returns the path portion for a designated URI. For example, if we specify the URI `/path/to/file.ext?=param=value`, and then use the command `URI::path`, this will return the following `/path/to/`.

Syntax

```
URI:path <uri>
```

```
URI::path <uri> depth
```

The `depth` option returns the path depth.

Example Use the following example to trigger the generation of a log message containing the path portion for a designated URI.

```
when HTTP_REQUEST {
set uri [HTTP::uri]
log "$uri path=[URI::path $uri] depth=[URI::path $uri depth]"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

URI::query

Description This command returns the query string portion for a designated URI. For example, if we specify the URI `/path/to/file.ext?=param=value`, the command `URI::path` returns `param=value`.

Syntax

```
URI::query <uri>
```

```
URI::query <uri> <param>
```

The <param> option returns the query parameter value that corresponds to the requested parameter name.

Example

Use the following example to trigger the generation of a log message containing the query parameter with value sent in the URI.

```
when HTTP_REQUEST {
set query [URI::query [HTTP::uri]]
log "The query portion of the URI is [HTTP::uri]: $query"
}
```

Valid Events

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_RESPONSE
- HTTP_RESPONSE_DATA

URL Commands

The following category commands is supported:

- URL::reputation

For information about aFleX commands, see aFleX Commands.

URL::reputation

Description This command returns the URL reputation values.

Syntax

```
URL::reputation <host> <require-web-category>
```

NOTE:

- It only supports numeric values to do the operations and the values returns a specific score (from 1-100).
- The `require-web-category` option is used to enable run-time-update. This option works with both HTTP/1.1 and HTTP/2 connections and is only applicable to HTTP_REQUEST and HTTP_REQUEST_DATA events.

Example Use the following example of an asynchronous web reputation lookup during an HTTP_REQUEST event is mentioned below:

```
when HTTP_REQUEST {
set host [HTTP::host]
log "Reputation : [URL::reputation $host require-web-
category]"
}
```

Use the following example of an asynchronous web reputation lookup during an HTTP_REQUEST_DATA event is mentioned below:

```
when HTTP_REQUEST {
HTTP::collect
}
when HTTP_REQUEST_DATA {
set host [HTTP::host]
log "Reputation : [URL::reputation $host require-web-
category]"
}
```

Valid Events

Valid with the following AAM events:

- AAM_AUTHENTICATION_INIT
- AAM_AUTHORIZATION_INIT
- AAM_AUTHORIZATION_CHECK
- AAM_RELAY_INIT

Valid with the following HTTP events:

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509 Commands

The following link commands are supported:

- X509::extensions
- X509::hash
- X509::issuer
- X509::not_valid_after
- X509::not_valid_before
- X509::serial_number
- X509::signature_algorithm
- X509::subject
- X509::subject_public_key
- X509::subject_public_key_RSA_bits
- X509::subject_public_key_type
- X509::text
- X509::verify_cert_error_string
- X509::version
- X509::whole

For information about aFleX commands, see aFleX Commands.

NOTE: In previous releases, these commands accepted certificates only in text format. The following X.509 commands now also accept certificates in Distinguished Encoding Rules (DER) format as input.

X509::extensions

Description This command returns the X.509 extensions set on the specified X.509 certificate. If an invalid certificate is supplied, a runtime TCL error is generated.

Syntax

```
X509::extensions <X509-certificate>
```

NOTE: `X509::extensions` returns a binary Byte array to preserve all the information.

Example Use this example for logging and inspecting X.509 certificate extensions from the client certificate during SSL client authentication.

```
when CLIENTSSL_CLIENTCERT {
    log "The X509 extensions for cert 0 are [X509::extensions
[SSL::cert 0]]."
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::hash

Description This command returns the MD5 (default) or SHA1 hash (fingerprint) of the specified X.509 certificate.

Syntax

```
X509::hash [md5|sha1|sha256] <X509 certificate>
```

NOTE: `X509::hash` no longer returns a text string but the actual hash value as a Byte array. To return a text string, use the `binary scan` command. See the Example 2 below.

Example Use the following example to log the hash for the specified certificate when complete client request header (method, URI, version, and all headers, not including the body) is parsed.

```
Example 1
when HTTP_REQUEST {
    log "The X509 hash for cert 0 is [X509::hash [SSL::cert
0]]."
}
Example 2
when CLIENTSSL_CLIENTCERT {
   set cert [SSL::cert 0]
   set sha256str ""
   log "This is the clientcert event"
   binary scan [X509::hash sha256 $cert] H* sha256str
   log "X509 Hash sha256: $sha256str"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::issuer

Description This command returns the issuer of the X.509 certificate.

Syntax

```
X509::issuer
```

Example Use the following example to log the certificate issuer when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 issuer for cert 0 is [X509::issuer
[SSL::cert 0]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE

- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::not_valid_after

Description This command returns the not-valid-after date of an X.509 certificate.

Syntax

```
X509::not_valid_after
```

Example Use the following example to log the date when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 is not valid after the date of cert 0
[X509::not_valid_after [SSL::cert 0]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::not_valid_before

Description This command returns the not-valid-before date of an X.509 certificate.

Syntax

```
X509::not_valid_before
```

Example Use the following example to log the date when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 is not valid before the date of cert 0
[X509::not_valid_before [SSL::cert 0]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::serial_number

Description This command returns the serial number of an X.509 certificate.

Syntax

```
X509::serial_number
```

Example Use the following example to log the serial number when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 serial number of cert 0 is [X509::serial_
number [SSL::cert 0]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA

- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::signature_algorithm

Description This command returns the signature algorithm of the specified X.509 certificate.

Syntax

```
X509::signature_algorithm <X509 certificate>
```

Example Use the following example to log the signature algorithm when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 signature algorithm of cert 0 is
[X509::signature_algorithm [SSL::cert 0]]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::subject

Description This command returns the subject of an X.509 certificate.

Syntax

```
X509::subject
```

Example Use the following example to set the certificate subject & log it when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
     set subject [X509::subject [SSL::cert 0]]
     log "The X509 subject of cert 0 is $subject."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::subject_public_key

Description This command returns the subject's public key of the specified X.509 certificate.

Syntax

```
X509::subject_public_key <X509 certificate>
```

Example Use the following example to log the public key when an SSL client certificate is received.

```
when CLIENTSSL_CLIENTCERT {
    log "The X509 subject public key for cert 0 is
[X509::subject_public_key [SSL::cert 0]]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::subject_public_key_RSA_bits

Description This command returns the size of the subject's public RSA key of an X.509 certificate. This command is only applicable when the public key type is RSA. Otherwise, the command generates an error.

Syntax

```
X509::subject_public_key_RSA_bits <X509 certificate>
```

Example Use the following example to log the public key size when an SSL client certificate is received.

```
when CLIENTSSL_CLIENTCERT {
    log "The X509 RSA public key size of cert 0 is
[X509::subject_public_key_RSA_bits [SSL::cert 0]]."
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::subject_public_key_type

Description This command returns the subject's public key type of the specified X.509 certificate. The returned value can be RSA, DSA, or unknown.

Syntax

```
X509::subject_public_key_type <X509 certificate>
```

Example Use the following example to log the Public Key Algorithm value under the Subject Public key info for the client certificate at level 0 sent by the client for authentication.

```
when CLIENTSSL_CLIENTCERT {
        log "x509 subject_public_key_type CLIENTSSL_
CLIENTCERT: [X509::subject_public_key_type [SSL::cert 0]]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::text

Description This command returns a certificate in human-readable (text) format.

Syntax

```
X509::text <X509 certificate>
```

Example Use the following example to log the human-readable certificate format when an SSL client certificate is received.

```
when CLIENTSSL_CLIENTCERT {
    log "The X509 readable text of cert 0 is [X509::text
[SSL::cert 0]]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

X509::verify_cert_error_string

Description This command returns the error string as an OpenSSL X.509 error string.

Syntax

```
X509::verify_cert_error_string
```

Example Use the following example to log the error string and the result code when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
    log "The X509 verify result of the peer is [X509::verify_
cert_error_string [SSL::verify_result]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::version

Description This command returns the version number of an X.509 certificate.

Syntax

```
X509::version
```

Example Use the following example to log the certificate version when an SSL handshake on the client side is completed.

```
when CLIENTSSL_HANDSHAKE {
     log "The X509 version of cert 0 is [X509::version
[SSL::cert 0]]."
}
```

Valid Events

- CLIENTSSL_CLIENTCERT
- CLIENTSSL_HANDSHAKE
- HTTP_REQUEST
- HTTP_REQUEST_DATA
- HTTP_REQUEST_SEND
- HTTP_RESPONSE
- HTTP_RESPONSE_CONTINUE
- HTTP_RESPONSE_DATA

X509::whole

Description This command returns the entire X.509 certificate in PEM format.

Syntax

```
X509::whole <X509 certificate>
```

Example Use the following example to log the whole client certificate at level 0 sent to ACOSÆ for client authentication in text form.

```
when CLIENTSSL_CLIENTCERT {
        log "x509 whole CLIENTSSL_CLIENTCERT: [X509::whole [SSL::cert
0]]"
}
```

Valid Events

All.

For information about aFleX events, see aFleX Events.

Deprecated and Disabled Commands

The aFleX scripting language previously supported some commands that are no longer supported. In addition, though aFleX is based on Tcl, many Tcl commands have been disabled for security reasons. See the following topics for lists of deprecated and disabled commands:

- Deprecated aFleX Commands
- Disabled Tcl Commands

For information about aFleX commands, see aFleX Commands.

Deprecated aFleX Commands

The global commands listed in Table 7 are deprecated. It is recommended not to use these commands. Instead, please use the recommended equivalent commands.

Table 7 : Deprecated Commands

Deprecated Command	Recommended Equivalent Command
client_addr	IP::client_addr
client_port	TCP::client_port or UDP::client_port
http_cookie	HTTP::cookie
http_header	HTTP::header
http_host	HTTP::host
http_method	HTTP::method
http_uri	HTTP::uri
http_version	HTTP::version
ip_protocol	IP::protocol
ip_ttl	IP::ttl
ip_tos	IP::tos
local_addr	IP::local_addr
redirect	HTTP::redirect
remote_addr	IP::remote_addr

Table 7 : Deprecated Commands

Deprecated Command	Recommended Equivalent Command
server_addr	IP::server_addr
server_port	TCP::server_port or UDP::server_port
use <command>	<command> This represents any valid and supported aFleX command. Please avoid use of "use" in front of any command. For the names of valid commands, see aFleX Commands.

For a list of disabled Tcl commands, see Disabled Tcl Commands.

Disabled Tcl Commands

For security, the following Tcl commands are disabled in the aFleX syntax. You cannot use these commands in aFleX scripts.

Table 8 : Disabled Tcl Commands

Disabled Tcl Commands			
after	exec	interp	seek
auto_execok	exit	load	socket
auto_import	fblocked	memory	source
auto_load	fconfigure	namespace	tcl_findLibrary
auto_mkindex	fcopy	open	tell
auto_mkindex_old	file	package	unknown
auto_qualify	fileevent	pid	update
auto_reset	filename	pkg::create	uplevel
bgerror	flush	pkg_mkIndex	upvar

Disabled Tcl Commands			
cd	gets	proc	vwait
close	glob	pwd	
eof	http	rename	

For a list of previously support aFleX commands that have been deprecated, see Deprecated aFleX Commands.

Contact Us