Troubleshoot and Test EEM Scripts

Contents

Introduction

This document describes Embedded Event Manager (EEM) script validation and introduces common operational considerations and failure scenarios.

Prerequisites

Requirements

This document assumes that the reader is already familiar with the Cisco IOS®/IOS XE® Embedded Event Manager (EEM) feature. If you are not already familiar with this feature, please read the EEM Feature Overview first.

EEM on the Catalyst 9K family of switches requires the DNA addon for the Network Essentials license level. Network Advantage fully supports EEM.

Components Used

The information in this document relates to EEM version 4.0 as implemented on the Catalyst family of switches.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

EEM is a useful feature when effectively deployed, but it is important to ensure that the EEM does precisely what the author intends. Poorly vetted scripts potentially lead to catastrophic problems in production. At best, the script performs in an undesired manner. This document provides useful information on how to test and verify EEM with CLI show commands, and also explains some common failure scenarios and the debugs used to identify and correct the problem.

EEM Validation with Show Commands

Confirm Timers are Active

When an EEM script is deployed that is triggered by a timer, if the script does not fire as expected, confirm that the timer is active and counts down.

Consider these EEM scripts named **test** and **test3**, respectively:

<#root>

```
event manager

applet test

 authorization bypass
event timer watchdog time 60
action 0010 syslog msg "Test script running"

event manager

applet test3

 authorization bypass
event timer watchdog name test3 time 300
action 0010 syslog msg "test3 script running"
```

- The first script (test) uses a 60 second (unnamed) watchdog timer to fire the script.
- The second script (test3) uses a 300 second watchdog timer named test3 to fire the script.

Configured timers and the current value of these timers can be viewed with the command **show event manager statistics server**.

Example

<#root>

```
Switch#

show event manager statistics server


EEM Queue Information
                        Triggered Dropped Queue Queue Average
Client                  Events    Events  Size  Max   Run Time
-----------------------------------------------------------------------------
Call Home               5         0       0     64    0.021
EEM Applets             181       0       0     64    0.003
EEM IOS .sh Scripts     0         0       0     128   0.000
EEM Tcl Scripts         0         0       0     64    0.000
```

```
iosp_global_eem_proc    30        0       0     16    0.004
onep event service init 0         0       0     128   0.000

EEM Policy Counters
Name Value
-------------------------------------------------------------------------------
```

```
EEM Policy Timers


Name                      Type

Time Remaining <-- EEM Countdown timer


-------------------------------------------------------------------------------

_EEMinternalname0

         watchdog              53.328

<--- Unnamed timers receive an internal name - this timer is for the 'test' policy


_EEMinternalname1         watchdog              37.120

test3

                     watchdog              183.232

<--- Named timers use their configured name - this is the named timed configured for policy 'test3'
```

Confirm Trigger Events are Firing

As discussed in the Confirm Timers are Active section of this document, IOS XE increments the Triggered Events column for the EEM Applets client row in the output of show event manager statistics server every time an EEM applet is fired. To verify your EEM script works as expected, perform your trigger event several times and examine the output of show event manager statistics server to confirm this value increments. If it does not, your script has not triggered.

When the command is run several times in sequence, the timer values to count down. When the timer reaches zero and the script runs, the triggered event count for EEM Applets also count up.

<#root>

```
Switch#

show event manager statistics server


EEM Queue Information


Triggered

 Dropped Queue Queue Average
Client

Events
```

```
    Events  Size  Max   Run Time
--------------------------------------------------------------------------------
Call Home              5         0       0     64    0.021
```

EEM Applets 183

```
       0       0     64    0.003
```

<--- "Triggered Events" column is incremented by 2 due to 2 timers firing

```
EEM IOS .sh Scripts     0         0       0     128   0.000
EEM Tcl Scripts         0         0       0     64    0.000
iosp_global_eem_proc    30        0       0     16    0.004
onep event service init 0         0       0     128   0.000

EEM Policy Counters
Name Value
--------------------------------------------------------------------------------


EEM Policy Timers
Name                           Type
```

Time Remaining

```
--------------------------------------------------------------------------------
```

_EEMinternalname0

```
              watchdog        56.215
_EEMinternalname1             watchdog        100.006
```

test3

```
                          watchdog        126.117
```

Note: If this does not occur, investigate your script to verify the configured timers.

Review Event History

For scripts that are not triggered by timers, the command show event manager history events is useful to confirm that applets are triggered as expected.

Consider this EEM script:

<#root>

```
event manager
```

applet test_manual

```
 authorization bypass
```

event none   <-- manual trigger type for testing

```
action 0010
```

syslog msg "I am a manually triggered script!" <-- message that is printed when script runs

This script runs when CLI event manager run test_manual is executed and prints a syslog message. Besides the output in syslog, the execution of this script can be verified by a review of the output of show event manager history events as shown:

<#root>

```
Switch#

show event manager history events


No. Job Id Proc Status   Time of Event

Event Type

           Name
1   5      Actv success  Fri Nov 6 15:45:07 2020

timer countdown



callback: Call Home process  <-- timer bases event that fired


2   18     Actv success  Mon Nov 9 14:12:33 2020   oir                  callback: Call Home process
3   19     Actv success  Mon Nov 9 14:12:40 2020   oir                  callback: Call Home process
4   20     Actv success  Fri Nov13 14:35:49 2020

none



applet: test_manual          <-- manually triggered event
```

EEM Validation with Manual Trigger

There are scenarios where it is desirable to manually trigger an EEM script, either to test the execution flow, or to perform a one-off action. This can be accomplished with an EEM script with a trigger of event none as demonstrated in this output:

<#root>

```
event manager

applet test_manual

 authorization bypass
event none
action 0010 syslog msg "I am a manually triggered script!"
```

Manually fire the script with the command **event manager run test_manual** from the enable prompt:

<#root>

```
Switch#
```

```
event manager run test_manual <-- Manually runs the script
```

```
Switch#
```

```
show log <-- Check for the log from action 10.
```

```
*Oct 26 21:24:40.762:
```

```
%HA_EM-6-LOG: test_manual: I am a manually triggered script!  <-- %HA_EM logs are from EEM events. The s
```

Operational Considerations

Ensure EEM scrips are validated prior to use in production. In general, there are a few primary ways a script fails to work as expected, three of which are discussed here.

This section shows how to check for these 3 common problems with EEM scripts:

1. CLI command failures: the command fails to parse and therefore fails to execute.
2. The script runs too long: EEM scripts have a default run time limit of 20 seconds. If this time is exceeded, the script stops before all commands run.
3. The script runs too often: Sometimes the trigger event used by the script can happen too frequently, which causes the script to rapidly fire. It is desirable to control how often and at what rate the script fires.

Problem: CLI Commands Fail to Execute

This example script contains several problems. It is a simple applet that appends the output of several show commands to a text file in local flash media:

<#root>

```
event manager
```

```
applet Data_Collection
```

```
 auth bypass
event timer
```

```
watchdog time 60
```

```
action 1.0 cli command "enable"
action 1.1 cli command "show clock | append flash:DataCollection.txt"
action 1.2 cli command "show interfaces breif | append flash:DataCollection.txt"
action 1.3 cli command "show ip route | append flash:Datacollection.txt"
action 1.4 cli command "show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt"
action 1.5 cli command "show platform hardware fed switch active qos stats internal cpu policer | appen
action 2.0 syslog msg  "Data Capture Complete"
```

The applet successfully ran, but did not generate the expected results:

<#root>

```
Switch#
```

show logging | in Capture

<-- Our script-generated syslog contains the string "Capture".

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete
```

<-- Action 2.0 successfully ran.

```
Switch#
```

dir flash: | in .txt

<-- We only expected one .txt file, however two appear in flash:

```
32792 -rw- 36 Mar 11 2021 20:40:01 +00:00 DataCollection.txt
32798 -rw- 807 Mar 11 2021 20:40:01 +00:00 Datacollection.txt
```

```
Switch#
```

more flash:DataCollection.txt

<-- the output of our expected .txt file is empty except for the output of "show clock

```
"
*20:40:01.343 UTC Thu Mar 11 2021
```

Use **debug embedded event manager action cli** to assist with applet verification.

<#root>

```
Switch#
```

debug embedded event manager action cli

```
*Mar 11 20:40:01.175: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_open called.
```

<-- The applet is called.

```
*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch>
*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch>enable
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show clock | appen
*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show interfaces br
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
```

show interfaces breif

```
 | append flash:DataCollection.txt
```

<-- Here is our first problem. "brief" is misspelled, so the command does not run.

```
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
```

^

```
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
```

% Invalid input detected at '^' marker. <-- CLI parser failure

```
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#
```

show ip route | append flash:Datacollection.txt <-- This created the second .txt file. The file name is

***Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#**

```
*Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#
```

show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt

<-- This problem is less intuitive.

```
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : CPU utilization for five
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : PID Runtime(ms) Invoked
```

the "exclude" argument reads everything beyond the pipe as the value that is to be excluded

```
.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 117 57246 448028 127 0.0
```

A problem like this will likely not be evident in debugging

```
.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 2 4488 16816 266 0.07% 0
```

This underscores the importance of pre-production testing to ensure the script performs as expected

```
.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 173 829 44093 18 0.07% 0
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 205 22271 1313739 16 0.0
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 467 238 2238 106 0.07% 0

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 81 12793 151345 84 0.07%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 232 22894 2621198 8 0.07
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 7 0 1 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 6 0 1 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 8 17 2804 6 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 9 33511 11402 2939 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 12 0 2 0 0.00% 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 10 106 1402 75 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 14 439 42047 10 0.00% 0.
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 11 0 1 0 0.00% 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 16 0 1 0 0.00% 0.00% 0.0
```

```
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 5 0 1 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 18 0 3 0 0.00% 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : 20+ lines read from cli,

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#
```

show platform hardware fed switch active qos stats internal cpu policer

```
 | append flash:DataCollection.txt
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : show platform hardware f
```

<-- Here, the syntax of the command was not properly parsed out before implementation. We are missing ar

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
```

^ <-- missing word queue

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
```

% Invalid input detected at '^' marker.   <-- CLI parser failure

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete
```

<-- The syslog from Action 2.0 writes.

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_close called.
```

<-- The applet closes out as expected after executing all configured actions.

Conclusion: Properly vet all EEM actions, and use debugs to proof against misconfigurations and typographical errors.

Problem: EEM Actions take Longer than the Maximum Runtime

In this scenario, a simple EEM is used to collect control-plane packet captures at 120-second intervals. It appends new capture data to an output file located in local storage media.

<#root>

```
event manager
```

applet Capture

```
event timer
```

watchdog time 120   <-- 120 second countdown timer

```
action 1.0 cli command "enable"
action 1.1 cli command "no monitor capture CPUCapture"
action 2.0 cli command "monitor capture CPUCapture control-plane in match any buffer circular"
action 2.1 cli command "monitor capture CPUCapture start"
action 3.0 wait 45
action 4.0 cli command "monitor capture CPUCapture stop"
action 4.1 cli command "show clock | append flash:CPUCapture.txt"
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"
```

```
action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"
```

You can easily determine that the EEM does not complete as expected. Check local logs for the syslog from action 5.0. This syslog prints upon each successful iteration of the applet. The log has not printed within the buffer and the file CPUCapture.txt was not written to flash:

<#root>

```
Switch#

show logging | include "CPUCapture Complete"



Switch#

dir flash: | include CPUCapture.txt
```

Enable debugs to investigate. The most commonly-used debug is **debug event manager action cli.** This utility prints a dialogue of the actions in sequence.

Debug output: The debug output shows the applet called successfully. The initial actions run without issue, but the capture fails to conclude.

<#root>

```
Switch#

debug event manager action cli



*Jan 28 22:55:54.742: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_open called.

<-- This is the initial message seen when the applet is called.


*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch>

The applet name can be seen within the line.


*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch>enable
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#no monitor capture CPU
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Capture does not exist
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT :
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:54.965: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCap
Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCap
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Started capture point : CPUCaptu
```

<-- The applet successfully creates and starts the capture.

```
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:56:15.187: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_close called.
```

<-- After 20 seconds, cli_close is called and the applet begins to exit.

```
*Jan 28 22:56:15.187: fh_server: fh_io_ipc_msg: received msg FH_MSG_CALLBACK_DONE from client 27 pclien
*Jan 28 22:56:15.187: fh_io_ipc_msg: EEM callback policy Capture has ended with abnormal exit status of
```

FF

```
*Jan 28 22:56:15.187:
```

EEM policy Capture has exceeded it's elapsed time limit of 20.0 seconds <-- We are informed that the pol

```
*Jan 28 22:56:15.187: fh_io_ipc_msg: received FH_MSG_API_CLOSE client=27
*Jan 28 22:56:15.187: tty is now going through its death sequence
```

*Note "

debug event manager all

" is used to enable all debugs related to event manager.

Solution: By default, EEM policies run no longer than 20 seconds. If the actions within the EEM take longer than 20 seconds to run, the EEM fails to complete. Ensure the runtime of your EEM is enough to allow for your applet actions to run. Configure **maxrun** to specify a more appropriate maximum runtime value.

Example

<#root>

```
event manager
```

applet Capture

```
event timer watchdog time 120
```

maxrun 60

<-- Maxrun 60 specifies the capture will run for a maximum of 60 seconds.

```
action 1.0 cli command "enable"
action 1.1 cli command "no monitor capture CPUCapture"
action 2.0 cli command "monitor capture CPUCapture control-plane in match any buffer circular"
action 2.1 cli command "monitor capture CPUCapture start"
action 3.0 wait 45
```

<-- The altered maxrun allows the capture to run for the necessary time.

```
action 4.0 cli command "monitor capture CPUCapture stop"
action 4.1 cli command "show clock | append flash:CPUCapture.txt"
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"
action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"
```

Problem: EEM Triggers too Often

Sometimes, several instances of a given trigger occur in a short amount of time. This could lead to excessive iterations of the applet and have serious consequences in the worst case.

This applet triggers on a particular syslog pattern, then gathers show command output and appends this output to a file. Specifically, the applet fires when line protocol drops for an identified interface:

<#root>

```
event manager
```

`applet MonitorLinkFlap`

```
event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"
action 1.0 cli command "enable"
action 1.1 cli command "show ip route | append flash:MonitorLinkFlap.txt "
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Moni
action 5.0 syslog msg "Link has flapped - Data gathered"
```

The applet fires each time the syslog is observed. An event such as an interface flap can occur rapidly in a short amount of time.

<#root>

```
Switch#
```

`sh log | in Data gathered`

```
*Jan 29 04:19:06.678: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

`<-- The applet generates this syslog each time it fires.`

```
*Jan 29 04:19:27.367: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:19:36.779: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:19:57.472: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:06.570: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:27.671: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:36.774: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:57.264: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

The applet ran numerous times over the course of a few minutes, which resulted in an undesirable output file with extraneous data. The file also continues to increase in size and continues to fill up local media. This simple example EEM does not pose much operational threat if ran repeatedly, but this scenario potentially leads to a crash with more complex scripts.

In this scenario, it would be beneficial to limit how often the applet is triggered.

Solution: Apply a rate limit to control how rapidly an applet runs. The **ratelimit** keyword is appended to the trigger statement and is associated with a value in seconds.

Example

```
<#root>

event manager

applet MonitorLinkFlap


event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"

ratelimit 60



<-- Ratelimit <seconds> specifies a minimum amount of time that must pass before the applet will again t


action 1.0 cli command "enable"
action 1.1 cli command "show clock | append flash:MonitorLinkFlap.txt "
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Moni
action 5.0 syslog msg "Link has flapped - Data gathered"
```

Related Information

Cisco IOS Embedded Event Manager 4.0

Best Practices and Useful Scripts for EEM