

EEM Python Module

Embedded Event Manager (EEM) policies support Python scripts. Python scripts can be executed as part of EEM actions in EEM applets.

- Prerequisites for the EEM Python Module, on page 1
- Information About EEM Python Module, on page 1
- How to Configure the EEM Python Policy, on page 4
- Additional References EEM Python Module, on page 9
- Feature Information for EEM Python Module, on page 10

Prerequisites for the EEM Python Module

Guest Shell must be working within the container. Guest Shell is not enabled by default. For more information see the *Guest Shell* feature.

Information About EEM Python Module

Python Scripting in EEM

Embedded Event Manager (EEM) policies support Python scripts. You can register Python scripts as EEM policies, and execute the registered Python scripts when a corresponding event occurs. The EEM Python script has the same event specification syntax as the EEM TCL policy.

Configured EEM policies run within the Guest Shell. Guest Shell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. The Guest Shell container provides a Python interpreter.

EEM Python Package

The EEM Python package can be imported to Python scripts for running EEM-specific extensions.

Note The EEM Python package is available only within the EEM Python script (The package can be registered with EEM, and has the EEM event specification in the first line of the script.) and not in the standard Python script (which is run using the Python script name).

The Python package includes the following application programming interfaces (APIs):

- Action APIs—Perform EEM actions and have default parameters.
- CLI-execution APIs—Run IOS commands, and return the output. The following are the list of CLI-execution APIs:
  - eem_cli_open()
  - eem_cli_exec()
  - eem_cli_read()
  - eem_cli_read_line()
  - eem_cli_run()
  - eem_cli_run_interactive()
  - eem_cli_read_pattern()
  - eem_cli_write()
  - eem_cli_close()
- Environment variables-accessing APIs—Get the list of built-in or user-defined variables. The following are the environment variables-accessing APIs:
  - eem_event_reqinfo ()-Returns the built-in variables list.
  - eem_user_variables()-Returns the current value of an argument.

Python-Supported EEM Actions

The Python package (is available only within the EEM script, and not available for the standard Python script) supports the following EEM actions:

- Syslog message printing
- Send SNMP traps
- Reload the box
- Switchover to the standby device
- Run a policy
- Track Object read
- Track Object Set

- Cisco Networking Services event generation

The EEM Python package exposes the interfaces for executing EEM actions. You can use the Python script to call these actions, and they are forwarded from the Python package via Cisco Plug N Play (PnP) to the action handler.

EEM Variables

An EEM policy can have the following types of variables:

- Event-specific built-in variables—A set of predefinied variables that are populated with details about the event that triggered the policy. The eem_event_reqinfo () API returns the builtin variables list. These variables can be stored in the local machine and used as local variables. Changes to local variables do not reflect in builtin variables.
- User-defined variables—Variables that can be defined and used in policies. The value of these variables can be referred in the Python script. While executing the script, ensure that the latest value of the variable is available. The eem_user_variables() API returns the current value of the argument that is provided in the API.

EEM CLI Library Command Extensions

The following CLI library commands are available within EEM for the Python script to work:

- eem_cli_close()—Closes the EXEC process and releases the VTY and the specified channel handler connected to the command.
- eem_cli_exec—Writes the command to the specified channel handler to execute the command. Then reads the output of the command from the channel and returns the output.
- eem_cli_open—Allocates a VTY, creates an EXEC CLI session, and connects the VTY to a channel handler. Returns an array including the channel handler.
- eem_cli_read()—Reads the command output from the specified CLI channel handler until the pattern of the device prompt occurs in the contents read. Returns all the contents read up to the match.
- eem_cli_read_line()—Reads one line of the command output from the specified CLI channel handler. Returns the line read.
- eem_cli_read_pattern()—Reads the command output from the specified CLI channel handler until the pattern that is to be matched occurs in the contents read. Returns all the contents read up to the match.
- eem_cli_run()—Iterates over the items in the *clist* and assumes that each one is a command to be executed in the enable mode. On success, returns the output of all executed commands and on failure, returns error.
- eem_cli_run_interactive()—Provides a sublist to the *clist* which has three items. On success, returns the output of all executed commands and on failure, returns the error. Also uses arrays when possible as a way of making things easier to read later by keeping expect and reply separated.
- eem_cli_write()—Writes the command that is to be executed to the specified CLI channel handler. The CLI channel handler executes the command.

How to Configure the EEM Python Policy

For the Python script to work, you must enable the Guest Shell. For more information, see the *Guest Shell* chapter.

Registering a Python Policy

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager directory user policy** *path*
4. **event manager policy** *policy-filename*
5. **exit**
6. **show event manager policy registered**
7. **show event manager history events**

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager directory user policy** *path* **Example:** `Device(config)# event manager directory user policy flash:/user_library`	Specifies a directory to use for storing user library files or user-defined EEM policies. **Note** You must have a policy in the specified path. For example, in this step, the eem_script.py policy is available in the flash:/user_library folder or path.
Step 4	**event manager policy** *policy-filename* **Example:** `Device(config)# event manager policy eem_script.py`	Registers a policy with EEM. • The policy is parsed based on the file extension. If the file extension is .py, the policy is registered as Python policy. • EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. When the **event manager policy** command is

	Command or Action	Purpose
		invoked, EEM examines the policy and registers it to be run when the specified event occurs.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 6	**show event manager policy registered** **Example:** `Device# show event manager policy registered`	Displays the registered EEM policies.
Step 7	**show event manager history events** **Example:** `Device# show event manager history events`	Displays EEM events that have been triggered.

Example

The following is sample output from the **show event manager policy registered** command:

```
Device# show event manager policy registered

No.  Class     Type    Event Type          Trap  Time Registered           Name
1    script    user    multiple            Off   Tue Aug 2 22:12:15 2016   multi_1.py
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi_1.py} sync {yes}
 trigger delay 10.000
  correlate event 1 or event 2
  attribute tag 1 occurs 1
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none


2    script    user    multiple            Off   Tue Aug 2 22:12:20 2016   multi_2.py
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi_2.py} sync {yes}
 trigger
  correlate event 1 or event 2
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none


3    script    user    multiple            Off   Tue Aug 2 22:13:31 2016   multi.tcl
 1: syslog: pattern {COUNTER}
 2: none: policyname {multi.tcl} sync {yes}
 trigger
  correlate event 1 or event 2
  attribute tag 1 occurs 1
 nice 0 queue-priority normal maxrun 100.000 scheduler rp_primary Secu none
```

Running Python Scripts as Part of EEM Applet Actions

Python Script: eem_script.py

An EEM applet can include a Python script with an action command. In this example, an user is trying to run a standard Python script as part of the EEM action, however; EEM Python package is not available in the standard Python script. The standard Python script in IOS has a package named *from cli import cli,clip* and this package can be used to execute IOS commands.

```
import sys
from cli import cli,clip,execute,executep,configure,configurep

intf= sys.argv[1:]
intf = ''.join(intf[0])


print ('This script is going to unshut interface %s and then print show ip interface
brief'%intf)

if intf == 'loopback55':
configurep(["interface loopback55","no shutdown","end"])
else :
cmd='int %s,no shut ,end' % intf
configurep(cmd.split(','))

executep('show ip interface brief')
```

This following is sample output from the **guestshell run python3** command.

```
Device#  guestshell run python3 /flash/eem_script.py loop55

This script is going to unshut interface loop55 and then print show ip interface brief
Line 1 SUCCESS: int loop55
Line 2 SUCCESS: no shut
Line 3 SUCCESS: end
Interface IP-Address OK? Method Status Protocol
Vlan1 unassigned YES NVRAM administratively down down
GigabitEthernet0/0 5.30.15.37 YES NVRAM up up
GigabitEthernet1/0/1 unassigned YES unset down down
GigabitEthernet1/0/2 unassigned YES unset down down
GigabitEthernet1/0/3 unassigned YES unset down down
GigabitEthernet1/0/4 unassigned YES unset up up
GigabitEthernet1/0/5 unassigned YES unset down down
GigabitEthernet1/0/6 unassigned YES unset down down
GigabitEthernet1/0/7 unassigned YES unset down down
GigabitEthernet1/0/8 unassigned YES unset down down
GigabitEthernet1/0/9 unassigned YES unset down down
GigabitEthernet1/0/10 unassigned YES unset down down
GigabitEthernet1/0/11 unassigned YES unset down down
GigabitEthernet1/0/12 unassigned YES unset down down
GigabitEthernet1/0/13 unassigned YES unset down down
GigabitEthernet1/0/14 unassigned YES unset down down
GigabitEthernet1/0/15 unassigned YES unset down down
GigabitEthernet1/0/16 unassigned YES unset down down
GigabitEthernet1/0/17 unassigned YES unset down down
GigabitEthernet1/0/18 unassigned YES unset down down
GigabitEthernet1/0/19 unassigned YES unset down down
GigabitEthernet1/0/20 unassigned YES unset down down
GigabitEthernet1/0/21 unassigned YES unset down down
GigabitEthernet1/0/22 unassigned YES unset down down
```

```
GigabitEthernet1/0/23 unassigned YES unset up up
GigabitEthernet1/0/24 unassigned YES unset down down
GigabitEthernet1/1/1 unassigned YES unset down down
GigabitEthernet1/1/2 unassigned YES unset down down
GigabitEthernet1/1/3 unassigned YES unset down down
GigabitEthernet1/1/4 unassigned YES unset down down
Te1/1/1 unassigned YES unset down down
Te1/1/2 unassigned YES unset down down
Te1/1/3 unassigned YES unset down down
Te1/1/4 unassigned YES unset down down
Loopback55 10.55.55.55 YES manual up up

Device#
Jun 7 12:51:20.549: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback55,
changed state to up
Jun 7 12:51:20.549: %LINK-3-UPDOWN: Interface Loopback55, changed state to up
```

The following is a sample script for printing messages to the syslog. This script must be stored in a file, copied to the file system on the device, and registered using the event manager policy file.

```
::cisco::eem::event_register_syslog tag "1"    pattern     COUNTER    maxrun 200

import eem
import time

eem.action_syslog("SAMPLE SYSLOG MESSAGE","6","TEST")
```

The following is sample script to print EEM environment variables. This script must be stored in a file, copied to the file system on the device, and registered using the event manager policy file.

```
::cisco::eem::event_register_syslog tag "1"    pattern     COUNTER    maxrun 200

import eem
import time

c = eem.env_reqinfo()


print "EEM Environment Variables"
for k,v in c.iteritems():
      print "KEY : " + k + str(" ---> ") + v

print "Built in Variables"
for i,j in a.iteritems() :
      print "KEY : " + i + str(" ---> ") + j
```

Adding a Python Script in an EEM Applet

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event** [**tag** *event-tag*] **syslog pattern** *regular-expression*
5. **action** *label* **cli command** *cli-string*

6. **action** *label* **cli command** *cli-string* [**pattern** *pattern-string*]
7. **end**
8. **show event manager policy active**
9. **show event manager history events**

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet interface_Shutdown`	Registers an applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**event** [**tag** *event-tag*] **syslog pattern** *regular-expression* **Example:** `Device(config-applet)# event syslog pattern "Interface Loopback55, changed state to administratively down"`	Specifies a regular expression to perform the syslog message pattern match.
Step 5	**action** *label* **cli command** *cli-string* **Example:** `Device(config-applet)# action 0.0 cli command "en"`	Specifies the IOS command to be executed when an EEM applet is triggered.
Step 6	**action** *label* **cli command** *cli-string* [**pattern** *pattern-string*] **Example:** `Device(config-applet)# action 1.0 cli command "guestshell run python3 /bootflash/eem_script.py loop55"`	Specifies the action to be specified with the **pattern** keyword. • Specify a regular expression pattern string that will match the next solicited prompt.
Step 7	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.
Step 8	**show event manager policy active** **Example:**	Displays EEM policies that are executing.

	Command or Action	Purpose
	`Device# show event manager policy active`	
Step 9	**show event manager history events** **Example:** `Device# show event manager history events`	Displays the EEM events that have been triggered.

What to do next

The following example shows how to trigger the Python script configured in the task:

```
Device(config)# interface loopback 55
Device(config-if)# shutdown
Device(config-if)# end
Device#

Mar 13 10:53:22.358 EDT: %SYS-5-CONFIG_I: Configured from console by console
Mar 13 10:53:24.156 EDT: %LINK-5-CHANGED:  Line protocol on Interface Loopback55, changed
state to down
Mar 13 10:53:27.319 EDT: %LINK-3-UPDOWN:  Interface Loopback55, changed state to
administratively down
Enter configuration commands, one per line.  End with CNTL/Z.
Mar 13 10:53:35.38 EDT:  %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback55, changed
 state to up
*Mar 13 10:53:35.39 EDT %LINK-3-UPDOWN: Interface Loopback55, changed state to up
+++ 10:54:33 edi37(default) exec +++
show ip interface br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   unassigned      YES unset  down                  down
GigabitEthernet0/0/1   unassigned      YES unset  down                  down
GigabitEthernet0/0/2   10.1.1.31       YES DHCP   up                    up
GigabitEthernet0/0/3   unassigned      YES unset  down                  down
GigabitEthernet0       192.0.2.1       YES manual up                    up
Loopback55             198.51.100.1    YES manual up                    up
Loopback66             172.16.0.1      YES manual up                    up
Loopback77             192.168.0.1     YES manual up                    up
Loopback88             203.0.113.1     YES manual up                    up
```

Additional References EEM Python Module

Related Documents

Related Topic	Document Title
EEM configuration	*Embedded Event Manager Configuration Guide*
EEM commands	*Embedded Event Manager Command Reference*
Guest Shell configuration	Guest Shell

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for EEM Python Module

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information for EEM Python Module

Feature Name	Release	Feature Information
EEM Python Module	Cisco IOS XE Everest 16.5.1a Cisco IOS XE Everest 16.5.1b	This feature supports Python scripts as EEM policies. No new commands were introduced. In Cisco IOS XE Everest 16.5.1a, this feature was implemented on the following platforms: • Cisco Catalyst 3650 Series Switches • Cisco Catalyst 3850 Series Switches • Cisco Catalyst 9300 Series Switches • In Cisco IOS XE Everest 16.5.1b, this feature was implemented on the following platforms: • Cisco ISR 4000 Series Integrated Service Routers
	Cisco IOS XE Everest 16.6.2	In Cisco IOS XE Everest 16.6.2, this feature was implemented on Cisco Catalyst 9400 Series Switches.
	Cisco IOS XE Fuji 16.8.1a	In Cisco IOS XE Fuji 16.8.1a, this feature was implemented on Cisco Catalyst 9500-High Performance Series Switches