Writing Embedded Event Manager Policies Using Tcl

This module describes how software developers can write and customize Embedded Event Manager (EEM) policies using Tool command language (Tcl) scripts to handle Cisco software faults and events. EEM is a policy-driven process by means of which faults in the Cisco software system are reported through a defined application programing interface (API). The EEM policy engine receives notifications when faults and other events occur. EEM policies implement recovery on the basis of the current state of the system and the actions specified in the policy for a given event. Recovery actions are triggered when the policy is run.

Prerequisites for Writing Embedded Event Manager Policies Using Tcl

- Before writing EEM policies, you should be familiar with the " Embedded Event Manager Overview " module.
- If you want to write EEM policies using the command-line interface (CLI) commands, you should be familiar with the " Writing Embedded Event Manager Policies Using the Cisco IOS CLI " module.

Information About Writing Embedded Event Manager Policies Using Tcl

EEM Policies

EEM offers the ability to monitor events and take informational or corrective action when the monitored events occur or reach a threshold. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs. There are two types of EEM policies: an applet or a script. An applet is a simple form of policy that is defined within the command-line interface (CLI) configuration. A script is a form of policy that is written in Tool Command Language (Tcl).

EEM Applet

An EEM applet is a concise method for defining event screening criteria and the actions to be taken when that event occurs. In EEM applet configuration mode, three types of configuration statements are supported. The event commands are used to specify the event criteria to trigger the applet to run, the action commands are used to specify an action to perform when the EEM applet is triggered, and the **set** command is used to set the value of an EEM applet variable. Currently only the _exit_status variable is supported for the **set** command.

Only one event configuration command is allowed within an applet configuration. When applet configuration submode is exited and no event command is present, a warning is displayed stating that no event is associated with the applet. If no event is specified, the applet is not considered registered. When no action is associated with the applet, events are still triggered but no actions are performed. Multiple action configuration commands are allowed within an applet configuration. Use the **show event manager policy registered** command to display a list of registered applets.

Before modifying an EEM applet, be aware that the existing applet is not replaced until you exit applet configuration mode. While you are in applet configuration mode modifying the applet, the existing applet may be executing. It is safe to modify the applet without unregistering it, because changes are written to a temporary file. When you exit applet configuration mode, the old applet is unregistered and the new version is registered.

Action configuration commands within an applet are uniquely identified using the *label* argument, which can be any string value. Actions are sorted within an applet in ascending alphanumeric key sequence using the *label* argument as the sort key, and they are run using this sequence. The same *label* argument can be used in different applets; the labels must be unique only within one applet.

The Embedded Event Manager schedules and runs policies on the basis of an event specification that is contained within the policy itself. When applet configuration mode is exited, EEM examines the event and action commands that are entered and registers the applet to be run when a specified event occurs.

For more details about writing EEM policies using the Cisco IOS CLI, see the “Writing Embedded Event Manager Policies Using the Cisco IOS CLI” module.

EEM Script

All Embedded Event Manager scripts are written in Tcl. Tcl is a string-based command language that is interpreted at run time. The version of Tcl supported is Tcl version 8.3.4 plus added script support. Scripts are defined using an ASCII editor on another device, not on the networking device. The script is then copied

to the networking device and registered with EEM. Tcl scripts are supported by EEM. As an enforced rule, Embedded Event Manager policies are short-lived run time routines that must be interpreted and executed in less than 20 seconds of elapsed time. If more than 20 seconds of elapsed time are required, the maxrun parameter may be specified in the event_register statement to specify any desired value.

EEM policies use the full range of the Tcl language's capabilities. However, Cisco provides enhancements to the Tcl language in the form of Tcl command extensions that facilitate the writing of EEM policies. The main categories of Tcl command extensions identify the detected event, the subsequent action, utility information, counter values, and system information.

EEM allows you to write and implement your own policies using Tcl. Writing an EEM script involves:

- Selecting the event Tcl command extension that establishes the criteria used to determine when the policy is run.
- Defining the event detector options associated with detecting the event.
- Choosing the actions to implement recovery or respond to the detected event.

EEM Policy Tcl Command Extension Categories

There are different categories of EEM policy Tcl command extensions.

Note The Tcl command extensions available in each of these categories for use in all EEM policies are described in later sections in this document.

Table 1: EEM Policy Tcl Command Extension Categories

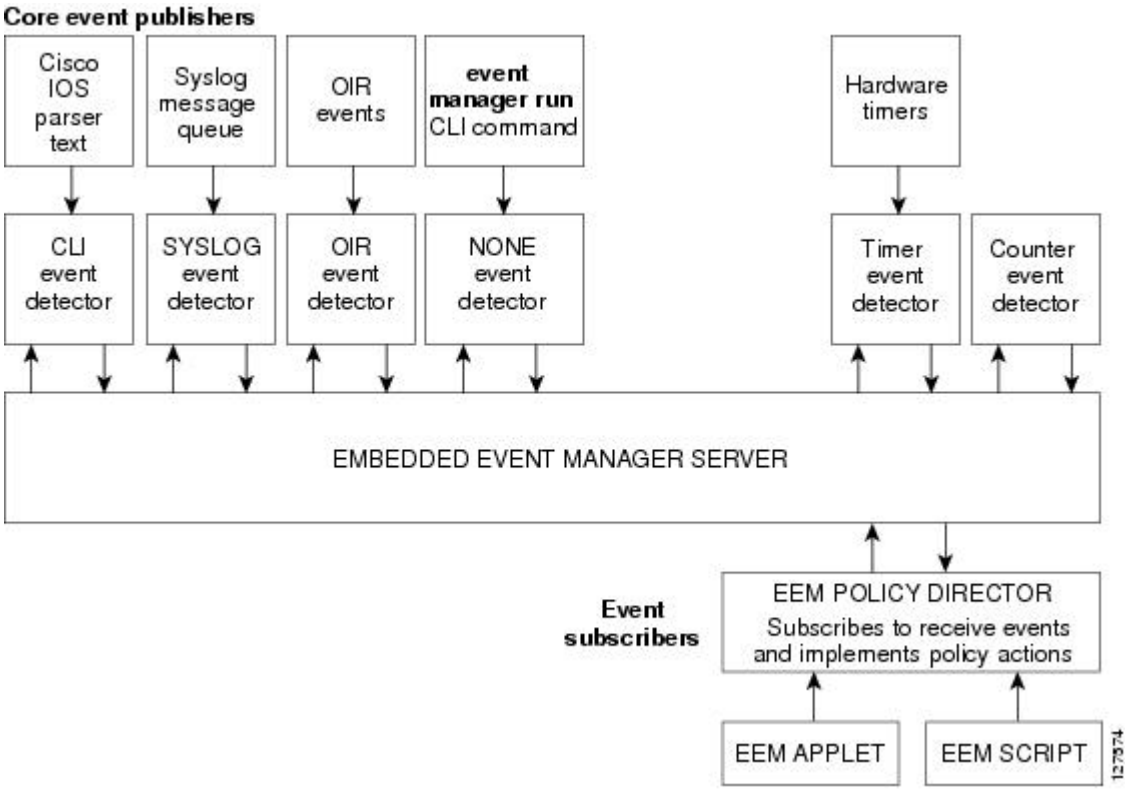
Category	Definition
EEM event Tcl command extensions (three types: event information, event registration, and event publish)	This category is represented by the **event_register_** *xxx* family of event-specific commands. There is a separate event information Tcl command extension in this category as well: **event_reqinfo**. This is the command used in policies to query the EEM for information about an event. There is also an EEM event publish Tcl command extension **event_publish**> *that publishes an application-specific event.*
EEM action Tcl command extensions	These Tcl command extensions (for example, **action_syslog**) are used by policies to respond to or recover from an event or fault. In addition to these extensions, developers can use the Tcl language to implement any action desired.
EEM utility Tcl command extensions	These Tcl command extensions are used to retrieve, save, set, or modify application information, counters, or timers.
EEM system information Tcl command extensions	This category is represented by the **sys_reqinfo _** *xxx* family of system-specific information commands. These commands are used by a policy to gather system information.
EEM context Tcl command extensions	These Tcl command extensions are used to store and retrieve a Tcl context (the visible variables and their values).

General Flow of EEM Event Detection and Recovery

EEM is a flexible, policy-driven framework that supports in-box monitoring of different components of the system with the help of software agents known as event detectors. The figure below shows the relationship between the EEM server, the core event publishers (event detectors), and the event subscribers (policies). Basically, event publishers screen events and publish them when there is a match on an event specification that is provided by the event subscriber. Event detectors notify the EEM server when an event of interest occurs.

When an event or fault is detected, Embedded Event Manager determines from the event publishers--an example would be the OIR events publisher in the figure below--if a registration for the encountered fault or event has occurred. EEM matches the event registration information with the event data itself. A policy registers for the detected event with the Tcl command extension **event_register _** *xxx*. The event information Tcl command extension **event_reqinfo** is used in the policy to query the Embedded Event Manager for information about the detected event.

Figure 1: Embedded Event Manager Core Event Detectors



Safe-Tcl

Safe-Tcl is a safety mechanism that allows untrusted Tcl scripts to run in an interpreter that was created in the safe mode. The safe interpreter has a restricted set of commands that prevent accessing some system resources and harming the host and other applications. For example, it does not allow commands to access critical Cisco IOS file system directories.

Cisco-defined scripts run in full Tcl mode, but user-defined scripts run in Safe-Tcl mode. Safe-Tcl allows Cisco to disable or customize individual Tcl commands. For more details about Tcl commands, go to http://www.tcl.tk/man/ .

The following list of Tcl commands are restricted with a few exceptions. Restrictions are noted against each command or command keyword:

- **cd** --Change directory is not allowed to one of the restricted Cisco directory names.
- **encoding** --The commands **encoding names**, **encoding convertfrom**, and **encoding convertto** are permitted. The **encoding system** command with no arguments is permitted, but the **encoding system** command with the **?encoding?** keyword is not permitted.
- **exec** --Not permitted.
- **fconfigure** --Permitted.
- **file** --The following are permitted:
  - **file dirname**
  - **file exists**
  - **file extension**
  - **file isdirectory**
  - **file join**
  - **file pathtype**
  - **file rootname**
  - **file split**
  - **file stat**
  - **file tail**
- **file** --The following are not permitted:
  - **file atime**
  - **file attributes**
  - **file channels**
  - **file copy**
  - **file delete**
  - **file executable**
  - **file isfile**
  - **file link**
  - **file lstat**
  - **file mkdir**
  - **file mtime**
  - **file nativename**
  - **file normalize**
  - **file owned**
  - **file readable**
  - **file readlink**
  - **file rename**
  - **file rootname**
  - **file separator**

  - **file size**
  - **file system**
  - **file type**
  - **file volumes**
  - **file writable**

- **glob** --The **glob** command is not permitted when searching in one of the restricted Cisco directories. Otherwise, it is permitted.
- **load** --Only files that are in the user policy directory or the user library directory are permitted to be loaded. Static packages (for example, libraries that consist of C code) are not permitted to be loaded with the **load** command.
- **open** --The **open** command is not allowed for a file that is located in one of the restricted Cisco directories.
- **pwd** --The **pwd** command is not permitted.
- **socket** --The **socket** command is permitted.
- **source** --The **source** command is permitted for files that are in the user policy directory or the user library directory.

Bytecode Support for EEM 2.4

EEM 2.4 introduces bytecode language (BCL) support by accepting files with the standard bytecode script extension .tbc. Tcl version 8.3.4 defines a BCL and includes a compiler that translates Tcl scripts into BCL. Valid EEM policy file extensions in EEM 2.4 for user and system policies are .tcl (Tcl Text files) and .tbc (Tcl bytecode files).

Storing Tcl scripts in bytecode improves the execution speed of the policy because the code is precompiled, creates a smaller policy size, and obscures the policy code. Obfuscation makes it a little more difficult to modify scripts and hides logic to preserve intellectual property rights.

Support for bytecode is being added to provide another option for release of supported and trusted code. We recommend that you only run well understood, or trusted and supported software on network devices. To generate Tcl bytecode for IOS EEM support, use TclPro versions 1.4 or 1.5.

To translate a Tcl script to bytecode you can use procomp, part of Free TclPro Compiler, or Active State Tcl Development Kit. When a Tcl script is compiled using procomp, the code is scrambled and a .tbc file is generated. The bytecode files are platform-independent and can be generated on any operating system on which TclPro is available, including Windows, Linux, and UNIX. Procomp is part of TclPro and available from http://www.tcl.tk/software/tclpro.

Registration Substitution

In addition to regular Tcl substitution, EEM 2.3 permits the substitution of an individual parameter in an EEM event registration statement line with an environment variable.

EEM 2.4 introduces the ability to replace multiple parameters in event registration statement lines with a single environment variable.

Note Only the first environment variable supports multiple parameter substitution. Individual parameters can still be specified with additional environment variables after the initial variable.

To illustrate the substitution, a single environment variable, $_eem_syslog_statement is configured as:

```
::cisco::eem::event_register_syslog pattern COUNT
```

Using the registration substitution, the $_eem_syslog_statement environment variable is used in the following EEM user policy:

```
$_eem_syslog_statement occurs $_eem_occurs_val
action_syslog "this is test 3"
```

Environment variables must be defined before a policy using them is registered. To define the $_eem_syslog_statement environment variable:

```
Device(config)# event manager environment eem_syslog_statement
::cisco::eem::event_register_syslog pattern COUNT
Device(config)# event manager environment eem_occurs_val 2
```

Cisco File Naming Convention for EEM

All Embedded Event Manager policy names, policy support files (for example, e-mail template files), and library filenames are consistent with the Cisco file naming convention. In this regard, Embedded Event Manager policy filenames adhere to the following specification:

- An optional prefix--Mandatory.--indicating, if present, that this is a system policy that should be registered automatically at boot time if it is not already registered. For example: Mandatory.sl_text.tcl.
- A filename body part containing a two-character abbreviation (see the table below) for the first event specified; an underscore part; and a descriptive field part that further identifies the policy.
- A filename suffix part defined as .tcl.

Embedded Event Manager e-mail template files consist of a filename prefix of email_template, followed by an abbreviation that identifies the usage of the e-mail template.

Embedded Event Manager library filenames consist of a filename body part containing the descriptive field that identifies the usage of the library, followed by _lib, and a filename suffix part defined as .tcl.

Table 2: Two-Character Abbreviation Specification

ap	event_register_appl
cl	event_register_cli
ct	event_register_counter
go	event_register_gold
if	event_register_interface
io	event_register_ioswdsysmon

la	event_register_ipsla
nf	event_register_nf
no	event_register_none
oi	event_register_oir
pr	event_register_process
rf	event_register_rf
rs	event_register_resource
rt	event_register_routing
rp	event_register_rpc
sl	event_register_syslog
sn	event_register_snmp
st	event_register_snmp_notification
so	event_register_snmp_object
tm	event_register_timer
tr	event_register_track
ts	event_register_timer_subscriber
wd	event_register_wdsysmon

How to Write Embedded Event Manager Policies Using Tcl

Registering and Defining an EEM Tcl Script

Perform this task to configure environment variables and register an EEM policy. EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. When an EEM policy is registered, the software examines the policy and registers it to be run when the specified event occurs.

Before you begin

You must have a policy available that is written in the Tcl scripting language. Sample policies are provided--see the details in the Sample EEM Policies, on page 19 to see which policies are available for the Cisco IOS release image that you are using--and these sample policies are stored in the system policy directory.

SUMMARY STEPS

1. **enable**
2. **show event manager environment** [**all**| *variable-name*]

3. **configure terminal**
4. **event manager environment** *variable-name string*
5. Repeat Step 4 to configure all the environment variables required by the policy to be registered in Step 6 .
6. **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]
7. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager environment** [**all**| *variable-name*] **Example:** `Device# show event manager environment all`	(Optional) Displays the name and value of EEM environment variables. • The optional **all** keyword displays all the EEM environment variables. • The optional *variable-name*argument displays information about the specified environment variable.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**event manager environment** *variable-name string* **Example:** `Device(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-6`	Configures the value of the specified EEM environment variable. • In this example, the software assigns a CRON timer environment variable to be set to the second minute of every hour of every day.
Step 5	Repeat Step 4 to configure all the environment variables required by the policy to be registered in Step 6 .	--
Step 6	**event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**] **Example:** `Device(config)# event manager policy tm_cli_cmd.tcl type system`	Registers the EEM policy to be run when the specified event defined within the policy occurs. • Use the **system** keyword to register a Cisco-defined system policy. • Use the **user** keyword to register a user-defined system policy. • Use the **trap** keyword to generate an SNMP trap when the policy is triggered.

	Command or Action	Purpose
		• In this example, the sample EEM policy named tm_cli_cmd.tcl is registered as a system policy.
Step 7	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

In the following example, the **show event manager environment** privileged EXEC command is used to display the name and value of all EEM environment variables.

```
Device# show event manager environment all
No.  Name                          Value
1    _cron_entry                   0-59/2 0-23/1 * * 0-6
2    _show_cmd                     show ver
3    _syslog_pattern               .*UPDOWN.*Ethernet1/0.*
4    _config_cmd1                  interface Ethernet1/0
5    _config_cmd2                  no shut
```

Displaying EEM Registered Policies

Perform this optional task to display EEM registered policies.

SUMMARY STEPS

1. **enable**
2. **show event manager policy registered** [**event-type** *event-name*] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*]

DETAILED STEPS

Step 1 **enable**

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **show event manager policy registered** [**event-type** *event-name*] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*]

Use this command with the **time-ordered** keyword to display information about currently registered policies sorted by time, for example:

Example:

```
Device# show event manager policy registered time-ordered
```

```
No.  Type    Event Type          Trap  Time Registered           Name
1    system  timer cron          Off   Wed May11  01:43:18 2005  tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240
2    system  syslog              Off   Wed May11  01:43:28 2005  sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90
3    system  proc abort          Off   Wed May11  01:43:38 2005  pr_cdp_abort.tcl
 instance 1 path {cdp2.iosproc}
 nice 0 priority normal maxrun 20
```

Use this command with the **name-ordered** keyword to display information about currently registered policies sorted by name, for example:

Example:

```
Device# show event manager policy registered name-ordered
No.  Type    Event Type          Trap  Time Registered           Name
1    system  proc abort          Off   Wed May11  01:43:38 2005  pr_cdp_abort.tcl
 instance 1 path {cdp2.iosproc}
 nice 0 priority normal maxrun 20
2    system  syslog              Off   Wed May11  01:43:28 2005  sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90
3    system  timer cron          Off   Wed May11  01:43:18 2005  tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240
```

Use this command with the **event-type** keyword to display information about currently registered policies for the event type specified in the *event-name* argument, for example:

Example:

```
Device# show event manager policy registered event-type syslog
No.  Type    Event Type          Time Registered           Name
1    system  syslog              Wed May11  01:43:28 2005  sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90
```

Unregistering EEM Policies

Perform this task to remove an EEM policy from the running configuration file. Execution of the policy is canceled.

SUMMARY STEPS

1. **enable**
2. **show event manager policy registered** [**event-type** *event-name*][**system**| **user**] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*]
3. **configure terminal**
4. **no event manager policy** *policy-filename*
5. **exit**
6. Repeat Unregistering EEM Policies to ensure that the policy has been removed.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy registered** [**event-type** *event-name*][**system**| **user**] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*] **Example:** `Device# show event manager policy registered`	(Optional) Displays the EEM policies that are currently registered. • The optional **system**or **user** keyword displays the registered system or user policies. • If no keywords are specified, EEM registered policies for all event types are displayed in time order.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**no event manager policy** *policy-filename* **Example:** `Device(config)# no event manager policy pr_cdp_abort.tcl`	Removes the EEM policy from the configuration, causing the policy to be unregistered. • In this example, the **no** form of the command is used to unregister a specified policy.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 6	Repeat Unregistering EEM Policies to ensure that the policy has been removed. **Example:** `Device# show event manager policy registered`	--

Examples

In the following example, the **show event manager policy registered** privileged EXEC command is used to display the three EEM policies that are currently registered:

```
Device# show event manager policy registered
No.  Type    Event Type          Trap  Time Registered           Name
1    system  timer cron          Off   Tue Oct11  01:43:18 2005 tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240.000
2    system  syslog              Off   Tue Oct11  01:43:28 2005 sl_intf_down.tcl
```

```
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90.000
3    system  proc abort          Off   Tue Oct11  01:43:38 2005 pr_cdp_abort.tcl
 instance 1 path {cdp2.iosproc}
 nice 0 priority normal maxrun 20.000
```

After the current policies are displayed, it is decided to delete the pr_cdp_abort.tcl policy using the **no** form of the **event manager policy** command:

```
Device# configure terminal
Device(config)# no event manager policy pr_cdp_abort.tcl
Device(config)# exit
```

The **show event manager policy registered** privileged EXEC command is entered again to display the EEM policies that are currently registered. The policy pr_cdp_abort.tcl is no longer registered.

```
Device# show event manager policy registered
No.  Type    Event Type          Trap  Time Registered           Name
1    system  timer cron          Off   Tue Oct11  01:45:17 2005 tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240.000
2    system  syslog              Off   Tue Oct11  01:45:27 2005 sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90.000
```

Suspending EEM Policy Execution

Perform this task to immediately suspend the execution of all EEM policies. Suspending policies, instead of unregistering them, might be necessary for reasons of temporary performance or security.

SUMMARY STEPS

1. **enable**
2. **show event manager policy registered** [**event-type** *event-name*][**system**| **user**] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*]
3. **configure terminal**
4. **event manager scheduler suspend**
5. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy registered** [**event-type** *event-name*][**system**| **user**] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*] **Example:**	(Optional) Displays the EEM policies that are currently registered. • The optional **system**or **user** keyword displays the registered system or user policies.

	Command or Action	Purpose
	`Device# show event manager policy registered`	• If no keywords are specified, EEM registered policies for all event types are displayed in time order.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**event manager scheduler suspend** **Example:** `Device(config)# event manager scheduler suspend`	Immediately suspends the execution of all EEM policies.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

In the following example, the **show event manager policy registered** privileged EXEC command is used to display all the EEM registered policies:

```
Device# show event manager policy registered
No.  Type    Event Type          Trap  Time Registered           Name
1    system  timer cron          Off   Sat Oct11  01:43:18 2003  tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240.000
2    system  syslog              Off   Sat Oct11  01:43:28 2003  sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90.000
3    system  proc abort          Off   Sat Oct11  01:43:38 2003  pr_cdp_abort.tcl
 instance 1 path {cdp2.iosproc}
 nice 0 priority normal maxrun 20.000
```

The **event manager scheduler suspend** command is entered to immediately suspend the execution of all EEM policies:

```
Device# configure terminal
Device(config)# event manager scheduler suspend
*Nov  2 15:34:39.000: %HA_EM-6-FMS_POLICY_EXEC: fh_io_msg: Policy execution has been
suspended
```

Managing EEM Policies

Perform this task to specify a directory to use for storing user library files or user-defined EEM policies.

Note This task applies only to EEM policies that are written using Tcl scripts.

SUMMARY STEPS

1. **enable**
2. **show event manager directory user** [**library**| **policy**]
3. **configure terminal**
4. **event manager directory user** {**library** *path*| **policy** *path*}
5. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager directory user** [**library**| **policy**] **Example:** `Device# show event manager directory user library`	(Optional) Displays the directory to use for storing EEM user library or policy files. • The optional **library**keyword displays the directory to use for user library files. • The optional **policy**keyword displays the directory to use for user-defined EEM policies.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 4	**event manager directory user** {**library** *path*| **policy** *path*}
	Example:
	`Device(config)# event manager directory user library disk0:/user_library`
	`Device(config)# event manager directory user library bootflash:/user_library`
		Specifies a directory to use for storing user library files or user-defined EEM policies.
		• Use the *path*argument to specify the absolute pathname to the user directory.

Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

In the following example, the **show event manager directory user** privileged EXEC command is used to display the directory, if it exists, to use for storing EEM user library files:

```
Device# show event manager directory user library
disk0:/user_library

Device# show event manager directory user library
bootflash:/user_library
```

Modifying History Table Size and Displaying EEM History Data

Perform this optional task to change the size of the history tables and to display EEM history data.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager history size** {**events** | **traps**} [*size*]
4. **exit**
5. **show event manager history events** [**detailed**] [**maximum** *number*]
6. **show event manager history traps** [**server** | **policy]**

DETAILED STEPS

Step 1 **enable**

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **configure terminal**

Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 3 **event manager history size** {**events** | **traps**} [*size*]

Use this command to change the size of the EEM event history table or the size of the EEM SNMP trap history table. In the following example, the size of the EEM event history table is changed to 30 entries:

Example:

```
Device(config)# event manager history size events 30
```

Step 4 **exit**

Exits global configuration mode and returns to privileged EXEC mode.

Example:

```
Device(config)# exit
```

Step 5 **show event manager history events** [**detailed**] [**maximum** *number*]

Use this command to display information about each EEM event that has been triggered.

Example:

```
Device# show event manager history events
No.  Time of Event             Event Type          Name
1    Fri Sep  9 13:48:40 2005  syslog              applet: one
2    Fri Sep  9 13:48:40 2005  syslog              applet: two
3    Fri Sep  9 13:48:40 2005  syslog              applet: three
4    Fri Sep  9 13:50:00 2005  timer cron          script: tm_cli_cmd.tcl
5    Fri Sep  9 13:51:00 2005  timer cron          script: tm_cli_cmd.tcl
```

Step 6 **show event manager history traps** [**server** | **policy**]

Use this command to display the EEM SNMP traps that have been sent either from the EEM server or from an EEM policy.

Example:

```
Device# show event manager history traps
No.  Time                      Trap Type           Name
1    Fri Sep  9 13:48:40 2005  server              applet: four
2    Fri Sep  9 13:57:03 2005  policy              script: no_snmp_test.tcl
```

Displaying Software Modularity Process Reliability Metrics Using EEM

Perform this optional task to display reliability metrics for Cisco IOS Software Modularity processes. The **show event manager metric processes** command is supported only in Software Modularity images.

SUMMARY STEPS

1. **enable**
2. **show event manager metric process** {**all**| *process-name*}

DETAILED STEPS

Step 1 **enable**

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **show event manager metric process** {**all**| *process-name*}

Use this command to display the reliability metric data for processes. The system keeps a record of when processes start and end, and this data is used as the basis for reliability analysis. In this partial example, the first and last entries showing the metric data for the processes on all the cards inserted in the system are displayed.

Example:

```
Device# show event manager metric process all
====================================
process name: devc-pty, instance: 1
sub_system id: 0, version: 00.00.0000
--------------------------------
last event type: process start
recent start time: Fri Oct10  20:34:40 2005
recent normal end time: n/a
recent abnormal end time: n/a
number of times started: 1
number of times ended normally: 0
number of times ended abnormally: 0
most recent 10 process start times:
--------------------------
Fri Oct10  20:34:40 2005
--------------------------
most recent 10 process end times and types:
cumulative process available time: 6 hours 30 minutes 7 seconds 378 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  0.100000000
number of abnormal ends within the past 60 minutes (since reload): 0
number of abnormal ends within the past 24 hours (since reload): 0
number of abnormal ends within the past 30 days (since reload): 0
.
.
.
====================================
process name: cdp2.iosproc, instance: 1
sub_system id: 0, version: 00.00.0000
--------------------------------
last event type: process start
recent start time: Fri Oct10  20:35:02 2005
recent normal end time: n/a
recent abnormal end time: n/a
number of times started: 1
number of times ended normally: 0
number of times ended abnormally: 0
most recent 10 process start times:
--------------------------
Fri Oct10  20:35:02 2005
--------------------------
most recent 10 process end times and types:

cumulative process available time: 6 hours 29 minutes 45 seconds 506 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  0.100000000
number of abnormal ends within the past 60 minutes (since reload): 0
number of abnormal ends within the past 24 hours (since reload): 0
number of abnormal ends within the past 30 days (since reload): 0
```

Troubleshooting Tips

Use the **debug event manager** command in privileged EXEC mode to troubleshoot EEM command operations. Use any debugging command with caution because the volume of output generated can slow or stop the device operations. We recommend that this command be used only under the supervision of a Cisco engineer.

Modifying the Sample EEM Policies

Perform this task to modify one of the sample policies. Cisco software contains some sample policies in the images that contain the Embedded Event Manager. Developers of EEM policies may modify these policies by customizing the event for which the policy is to be run and the options associated with logging and responding to the event. In addition, developers may select the actions to be implemented when the policy runs.

Sample EEM Policies

Cisco includes a set of sample policies shown in the table below. You can copy the sample policies to a user directory and then modify the policies, or you can write your own policies. Tcl is currently the only Cisco-supported scripting language for policy creation. Tcl policies can be modified using a text editor such as Emacs. Policies must execute within a defined number of seconds of elapsed time, and the time variable can be configured within a policy. The default is currently 20 seconds.

The table below describes the sample EEM policies.

Table 3: Sample EEM Policy Descriptions

Name of Policy	Description
pr_cdp_abort.tcl	Introduced with Cisco Software Modularity images. This policy monitors for cdp2.iosproc process abort events. It will log a message to SYSLOG and send an e-mail with the details of the abort.
pr_crash_reporter.tcl	Introduced with Cisco Software Modularity images. This policy monitors for all process abort events. When an event occurs, the policy will send crash information, including the crashdump file, to the specified URL where a CGI script processes the data.
pr_iprouting_abort.tcl	Introduced with Cisco Software Modularity images. This policy monitors for iprouting.iosproc process abort events. It will log a message to SYSLOG and send an e-mail with the details of the abort.
sl_intf_down.tcl	This policy runs when a configurable syslog message is logged. It will execute a configurable CLI command and e-mail the results.
tm_cli_cmd.tcl	This policy runs using a configurable CRON entry. It will execute a configurable CLI command and e-mail the results.
tm_crash_history.tcl	Introduced with Cisco Software Modularity images. This policy runs at midnight every day and e-mails a process crash history report to a specified e-mail address.

Name of Policy	Description
tm_crash_reporter.tcl	This policy runs 5 seconds after it is registered. If the policy is saved in the configuration, it will also run each time that the device is reloaded. The policy will prompt for the reload reason. If the reload was due to a crash, the policy will search for the latest crashinfo file and send this information to a specified URL location.
tm_fsys_usage.tcl	Introduced with Cisco Software Modularity images. This policy runs using a configurable CRON entry and monitors disk space usage. A syslog message will be displayed if disk space usage crosses configurable thresholds.
wd_mem_reporter.tcl	Introduced with Cisco Software Modularity images. This policy reports on low system memory conditions when the amount of memory available falls below 20 percent of the initial available system memory. A syslog message will be displayed and, optionally, an e-mail will be sent.

For more details about the sample policies available and how to run them, see the EEM Event Detector Demo Examples, on page 37.

SUMMARY STEPS

1. **enable**
2. **show event manager policy available detailed** *policy-filename*
3. Cut and paste the contents of the sample policy displayed on the screen to a text editor.
4. Edit the policy and save it with a new filename.
5. Copy the new file back to the device flash memory.
6. **configure terminal**
7. **event manager directory user** {**library** *path*| **policy** *path*}
8. **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]

DETAILED STEPS

Step 1 enable

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 show event manager policy available detailed *policy-filename*

Displays the actual specified sample policy including details about the environment variables used by the policy and instructions for running the policy. The **detailed** keyword was introduced for the **show event manager policy available** and the **show event manager policy registered** commands. Depending on your release, you may need to copy one of the two Tcl scripts from the configuration examples section in this document (see the Programming Policies with Tcl Sample Scripts Example, on page 45). In the following example, details about the sample policy tm_cli_cmd.tcl are displayed on the screen.

Example:

```
Device# show event manager policy available detailed tm_cli_cmd.tcl
```

Step 3 Cut and paste the contents of the sample policy displayed on the screen to a text editor.

Use the edit and copy functions to move the contents from the device to a text editor on another device.

Step 4 Edit the policy and save it with a new filename.

Use the text editor to modify the policy as a Tcl script. For file naming conventions, see the Cisco File Naming Convention for EEM, on page 7.

Step 5 Copy the new file back to the device flash memory.

Copy the file to the flash file system on the device--typically disk0:. For more details about copying files, see the "Using the Cisco IOS File System" chapter in the *Configuration Fundamentals Configuration Guide*.

Copy the file to the flash file system on the device--typically bootflash:. For more details about copying files, see the "Using the Cisco IOS File System" chapter in the *Configuration Fundamentals Configuration Guide*.

Step 6 **configure terminal**

Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 7 **event manager directory user** {**library** *path*| **policy** *path*}

Specifies a directory to use for storing user library files or user-defined EEM policies. In the following example, the user_library directory on disk0 is specified as the directory for storing user library files.

Specifies a directory to use for storing user library files or user-defined EEM policies. In the following example, the user_library directory on bootflash is specified as the directory for storing user library files.

Example:

```
Device(config)# event manager directory user library disk0:/user_library

Device(config)# event manager directory user library bootflash:/user_library
```

Step 8 **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]

Registers the EEM policy to be run when the specified event defined within the policy occurs. In the following example, the new EEM policy named test.tcl is registered as a user-defined policy.

Example:

```
Device(config)# event manager policy test.tcl type user
```

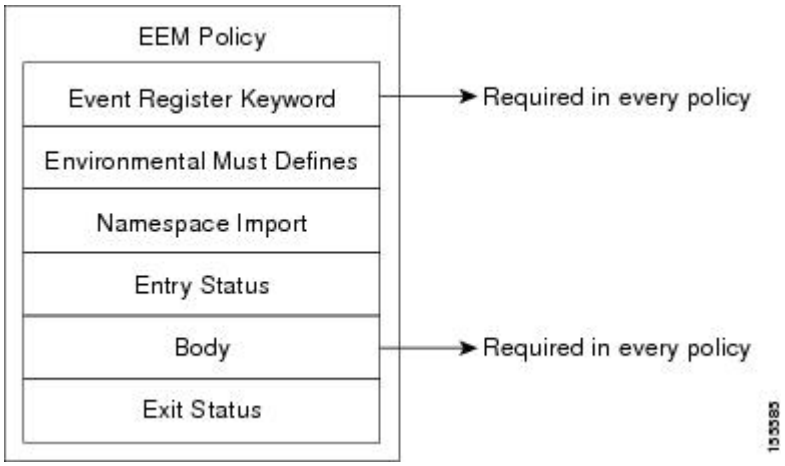
Programming EEM Policies with Tcl

Perform this task to help you program a policy using Tcl command extensions. We recommend that you copy an existing policy and modify it. There are two required parts that must exist in an EEM Tcl policy: the **event_register** Tcl command extension and the body. All other sections shown in the Tcl Policy Structure and Requirements, on page 22 concept are optional.

Tcl Policy Structure and Requirements

All EEM policies share the same structure, shown in the figure below. There are two parts of an EEM policy that are required: the **event_register** Tcl command extension and the body. The remaining parts of the policy are optional: environment must defines, namespace import, entry status, and exit status.

Figure 2: Tcl Policy Structure and Requirements



The start of every policy must describe and register the event to detect using an **event_register** Tcl command extension. This part of the policy schedules the running of the policy . The following example Tcl code shows how to register the **event_register_timer** Tcl command extension:

```
::cisco::eem::event_register_timer cron name crontimer2 cron_entry $_cron_entry maxrun 240
```

The environment must defines section is optional and includes the definition of environment variables. The following example Tcl code shows how to check for, and define, some environment variables.

```
# Check if all the env variables that we need exist.
# If any of them does not exist, print out an error msg and quit.
if {![info exists _email_server]} {
    set result \
      "Policy cannot be run: variable _email_server has not been set"
    error $result $errorInfo
}
if {![info exists _email_from]} {
    set result \
      "Policy cannot be run: variable _email_from has not been set"
    error $result $errorInfo
}
if {![info exists _email_to]} {
    set result \
      "Policy cannot be run: variable _email_to has not been set"
    error $result $errorInfo
```

The namespace import section is optional and defines code libraries. The following example Tcl code shows how to configure a namespace import section.

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
```

The body of the policy is a required structure and might contain the following:

- The **event_reqinfo**event information Tcl command extension that is used to query the EEM for information about the detected event.
- The action Tcl command extensions, such as **action_syslog**, that are used to specify EEM specific actions.
- The system information Tcl command extensions, such as **sys_reqinfo_routername**, that are used to obtain general system information.
- Use of the SMTP library (to send e-mail notifications) or the CLI library (to run CLI commands) from a policy.
- The **context_save** and **context_retrieve** Tcl command extensions that are used to save Tcl variables for use by other policies.

The following example Tcl code shows the code to query an event and log a message as part of the body section.

```
# Query the event info and log a message.
array set arr_einfo [event_reqinfo]

if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

global timer_type timer_time_sec
set timer_type $arr_einfo(timer_type)
set timer_time_sec $arr_einfo(timer_time_sec)

# Log a message.
set msg [format "timer event: timer type %s, time expired %s" \
        $timer_type [clock format $timer_time_sec]]

action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
      $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

EEM Entry Status

The entry status part of an EEM policy is used to determine if a prior policy has been run for the same event, and to determine the exit status of the prior policy. If the _entry_status variable is defined, a prior policy has already run for this event. The value of the _entry_status variable determines the return code of the prior policy.

Entry status designations may use one of three possible values: 0 (previous policy was successful), Not=0 (previous policy failed), and Undefined (no previous policy was executed).

EEM Exit Status

When a policy finishes running its code, an exit value is set. The exit value is used by the Embedded Event Manager to determine whether or not to apply the default action for this event, if any. A value of zero means do not perform the default action. A value of nonzero means perform the default action. The exit status will be passed to subsequent policies that are run for the same event.

EEM Policies and Cisco Error Number

Some EEM Tcl command extensions set a Cisco Error Number Tcl global variable _cerrno. Whenever _cerrno is set, four other Tcl global variables are derived from _cerrno and are set along with it (_cerr_sub_num, _cerr_sub_err, _cerr_posix_err, and _cerr_str).

For example, the **action_syslog** command in the example below sets these global variables as a side effect of the command execution:

```
action_syslog priority warning msg "A sample message generated by action_syslog"
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

_cerrno: 32-Bit Error Return Values

The _cerrno set by a command can be represented as a 32-bit integer of the following form:

```
XYSSSSSSSSSSSSSEEEEEEEEPPPPPPPPP
```

For example, the following error return value might be returned from an EEM Tcl command extension:

```
862439AE
```

This number is interpreted as the following 32-bit value:

```
10000110001001000011100110101110
```

This 32-bit integer is divided up into the five variables shown in the table below.

Table 4: _cerrno: 32-Bit Error Return Value Variables

Variable	Description
XY	The error class (indicates the severity of the error). This variable corresponds to the first two bits in the 32-bit error return value; 10 in the case above, which indicates CERR_CLASS_WARNING: See the table below for the four possible error class encodings specific to this variable.
SSSSSSSSSSSSS	The subsystem number that generated the most recent error (13 bits = 8192 values). This is the next 13 bits of the 32-bit sequence, and its integer value is contained in $_cerr_sub_num.
Variable	Description
EEEEEEEE	The subsystem specific error number (8 bits = 256 values). This segment is the next 8 bits of the 32-bit sequence, and the string corresponding to this error number is contained in $_cerr_sub_err.
PPPPPPPP	The pass-through POSIX error code (9 bits = 512 values). This represents the last of the 32-bit sequence, and the string corresponding to this error code is contained in $_cerr_posix_err.

Error Class Encodings for XY

The first variable, XY, references the possible error class encodings shown in the table below.

Table 5: Error Class Encodings

00	CERR_CLASS_SUCCESS
01	CERR_CLASS_INFO
10	CERR_CLASS_WARNING
11	CERR_CLASS_FATAL

An error return value of zero means SUCCESS.

SUMMARY STEPS

1. **enable**
2. **show event manager policy available detailed** *policy-filename*
3. Cut and paste the contents of the sample policy displayed on the screen to a text editor.
4. Define the required **event_register** Tcl command extension.
5. Add the appropriate namespace under the ::cisco hierarchy.
6. Program the must defines section to check for each environment variable that is used in this policy.
7. Program the body of the script.
8. Check the entry status to determine if a policy has previously run for this event.
9. Check the exit status to determine whether or not to apply the default action for this event, if a default action exists.
10. Set Cisco Error Number (_cerrno) Tcl global variables.
11. Save the Tcl script with a new filename, and copy the Tcl script to the device.
12. **configure terminal**
13. **event manager directory user** {**library** *path*| **policy** *path*}
14. **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]
15. Cause the policy to execute, and observe the policy.
16. Use debugging techniques if the policy does not execute correctly.

DETAILED STEPS

Step 1 enable

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 show event manager policy available detailed *policy-filename*

Displays the actual specified sample policy including details about the environment variables used by the policy and instructions for running the policy. The **detailed** keyword was introduced for the **show event manager policy available** and the **show event manager policy registered** commands. Depending on your release, you must copy one of the two Tcl scripts from the configuration examples section in this document (see the Programming Policies with Tcl Sample

Scripts Example, on page 45). In the following example, details about the sample policy tm_cli_cmd.tcl are displayed on the screen.

Example:

```
Device# show event manager policy available detailed tm_cli_cmd.tcl
```

Step 3 Cut and paste the contents of the sample policy displayed on the screen to a text editor.

Use the edit and copy functions to move the contents from the device to a text editor on another device. Use the text editor to edit the policy as a Tcl script.

Step 4 Define the required **event_register** Tcl command extension.

Choose the appropriate **event_register** Tcl command extension from the table below for the event that you want to detect, and add it to the policy.

Table 6: EEM Event Registration Tcl Command Extensions

Event Registration Tcl Command Extensions
event_register_appl
event_register_cli
event_register_counter
event_register_gold
event_register_interface
event_register_ioswdsysmon
event_register_ipsla
event_register_nf
event_register_none
event_register_oir
event_register_process
event_register_resource
event_register_rf
event_register_routing
event_register_rpc
event_register_snmp
event_register_snmp_notification
event_register_snmp_object

Event Registration Tcl Command Extensions
event_register_syslog
event_register_timer
event_register_timer_subscriber
event_register_track
event_register_wdsysmon

Step 5 Add the appropriate namespace under the ::cisco hierarchy.

Policy developers can use the new namespace ::cisco in Tcl policies in order to group all the extensions used by Cisco IOS EEM. There are two namespaces under the ::cisco hierarchy, and the table below shows which category of EEM Tcl command extension belongs under each namespace.

Table 7: Cisco IOS EEM Namespace Groupings

Namespace	Category of Tcl Command Extension
::cisco::eem	EEM event registration
	EEM event information
	EEM event publish
	EEM action
	EEM utility
	EEM context library
	EEM system information
	CLI library
::cisco::lib	SMTP library

Note Make sure that you import the appropriate namespaces or use the qualified command names when using the above commands.

Step 6 Program the must defines section to check for each environment variable that is used in this policy.

This is an optional step. Must defines are a section of the policy that tests whether any EEM environment variables that are required by the policy are defined before the recovery actions are taken. The must defines section is not required if the policy does not use any EEM environment variables. EEM environment variables for EEM scripts are Tcl global variables that are defined external to the policy before the policy is run. To define an EEM environment variable, use the Embedded Event Manager configuration command **event manager environment** CLI command. By convention all Cisco EEM environment variables begin with " _ " (an underscore). In order to avoid future conflict, customers are urged not to define new variables that start with " _ ".

Note You can display the Embedded Event Manager environment variables set on your system by using the **show event manager environment** privileged EXEC command.

For example, Embedded Event Manager environment variables defined by the sample policies include e-mail variables. The sample policies that send e-mail must have the variables shown in the table below set in order to function properly.

The table below describes the e-mail-specific environment variables used in the sample EEM policies.

Table 8: E-mail-Specific Environmental Variables Used by the Sample Policies

Environment Variable	Description	Example
_email_server	A Simple Mail Transfer Protocol (SMTP) mail server used to send e-mail.	The e-mail server name can be in any one of the following template formats: • username:password@host • username@host • host
_email_to	The address to which e-mail is sent.	engineering@example.com
_email_from	The address from which e-mail is sent.	devtest@example.com
_email_cc	The address to which the e-mail must be copied.	manager@example.com

The following example of a must define section shows how to program a check for e-mail-specific environment variables.

Example of Must Defines

Example:

```
if {![info exists _email_server]} {
    set result \
        "Policy cannot be run: variable _email_server has not been set"
    error $result $errorInfo
}
if {![info exists _email_from]} {
    set result \
        "Policy cannot be run: variable _email_from has not been set"
    error $result $errorInfo
}
if {![info exists _email_to]} {
    set result \
        "Policy cannot be run: variable _email_to has not been set"
    error $result $errorInfo
}
if {![info exists _email_cc]} {
    set result \
        "Policy cannot be run: variable _email_cc has not been set"
    error $result $errorInfo
}
```

Step 7 Program the body of the script.

In this section of the script, you can define any of the following:

- The **event_reqinfo**event information Tcl command extension that is used to query the EEM for information about the detected event.
- The action Tcl command extensions, such as **action_syslog**, that are used to specify EEM specific actions.

- The system information Tcl command extensions, such as **sys_reqinfo_routername**, that are used to obtain general system information.
- The **context_save** and **context_retrieve** Tcl command extensions that are used to save Tcl variables for use by other policies.
- Use of the SMTP library (to send e-mail notifications) or the CLI library (to run CLI commands) from a policy.

Step 8 Check the entry status to determine if a policy has previously run for this event.

If the prior policy is successful, the current policy may or may not require execution. Entry status designations may use one of three possible values: 0 (previous policy was successful), Not=0 (previous policy failed), and Undefined (no previous policy was executed).

Step 9 Check the exit status to determine whether or not to apply the default action for this event, if a default action exists.

A value of zero means do not perform the default action. A value of nonzero means perform the default action. The exit status will be passed to subsequent policies that are run for the same event.

Step 10 Set Cisco Error Number (_cerrno) Tcl global variables.

Some EEM Tcl command extensions set a Cisco Error Number Tcl global variable _cerrno. Whenever _cerrno is set, four other Tcl global variables are derived from _cerrno and are set along with it (_cerr_sub_num, _cerr_sub_err, _cerr_posix_err, and _cerr_str).

For example, the **action_syslog** command in the example below sets these global variables as a side effect of the command execution:

Example:

```
action_syslog priority warning msg "A sample message generated by action_syslog
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

Step 11 Save the Tcl script with a new filename, and copy the Tcl script to the device.

Embedded Event Manager policy filenames adhere to the following specification:

- An optional prefix--Mandatory.--indicating, if present, that this is a system policy that should be registered automatically at boot time if it is not already registered. For example: Mandatory.sl_text.tcl.
- A filename body part containing a two-character abbreviation (see EEM Policies and Cisco Error Number, on page 24) for the first event specified; an underscore character part; and a descriptive field part further identifying the policy.
- A filename suffix part defined as .tcl.

For more details, see the Cisco File Naming Convention for EEM, on page 7.

Copy the file to the flash file system on the device--typically disk0:. For more details about copying files, see the "Using the Cisco IOS File System" chapter in the Cisco IOS Configuration Fundamentals Configuration Guide .

Copy the file to the flash file system on the device--typically bootflash:. For more details about copying files, see the "Using the Cisco IOS File System" chapter in the Cisco IOS Configuration Fundamentals Configuration Guide .

Step 12 **configure terminal**

Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 13 **event manager directory user** {**library** *path*| **policy** *path*}

Specifies a directory to use for storing user library files or user-defined EEM policies. In the following example, the user_library directory on disk0 is specified as the directory for storing user library files.

Specifies a directory to use for storing user library files or user-defined EEM policies. In the following example, the user_library directory on bootflash is specified as the directory for storing user library files.

Example:

```
Device(config)# event manager directory user library disk0:/user_library
```

```
Device(config)# event manager directory user library bootflash:/user_library
```

Step 14 **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]

Registers the EEM policy to be run when the specified event defined within the policy occurs. In the following example, the new EEM policy named cl_mytest.tcl is registered as a user-defined policy.

Example:

```
Device(config)# event manager policy cl_mytest.tcl type user
```

Step 15 Cause the policy to execute, and observe the policy.

To test that the policy runs, generate the conditions that will cause the policy to execute and observe that the policy runs as expected.

Step 16 Use debugging techniques if the policy does not execute correctly.

Use the Cisco IOS **debug event manager** CLI command with its various keywords to debug issues. Refer to the Troubleshooting Tips, on page 30 for details about using Tcl-specific keywords.

Troubleshooting Tips

- Use the **debug event manager tcl commands** CLI command to debug issues with Tcl extension commands. When enabled, this command displays all data that is passed in and read back from the TTY session that handles the CLI interactions. This data helps ensure users that the commands they are passing to the CLI are valid.
- The CLI library allows users to run CLI commands and obtain the output of commands in Tcl. Use the **debug event manager tcl cli-library** CLI command to debug issues with the CLI library.
- The SMTP library allows users to send e-mail messages to an SMTP e-mail server. Use the **debug event manager tcl smtp_library** CLI command to debug issues with the SMTP library. When enabled, this command displays all data that is passed in and read back from the SMTP library routines. This data helps ensure users that the commands they are passing to the SMTP library are valid.

- Tcl is a flexible language that allows you to override commands. For example, you can modify the **set** command and create a version of the **set** command that displays a message when a scalar variable is set. When the **set** command is entered in a policy, a message is displayed anytime a scalar variable is set, and this provides a way to debug scalar variables. To view an example of this debugging technique, see the Tracing Tcl set Command Operations Example, on page 56.

To view examples of the some of these debugging techniques, see the Debugging Embedded Event Manager Policies Examples, on page 54.

Creating an EEM User Tcl Library Index

Perform this task to create an index file that contains a directory of all the procedures contained in a library of Tcl files. This task allows you to test library support in EEM Tcl. In this task, a library directory is created to contain the Tcl library files, the files are copied into the directory, and an index tclIndex) is created that contains a directory of all the procedures in the library files. If the index is not created, the Tcl procedures will not be found when an EEM policy is run that references a Tcl procedure.

SUMMARY STEPS

1. On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl library files into the directory.
2. **tclsh**
3. **auto_mkindex** *directory_name* ***.tcl**
4. Copy the Tcl library files and the tclIndex file to the directory used for storing user library files on the target device.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device.
6. **enable**
7. **configure terminal**
8. **event manager directory user library** *path*
9. **event manager directory user policy** *path*
10. **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
11. **event manager run** *policy-name*

DETAILED STEPS

Step 1 On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl library files into the directory.

The following example files can be used to create a tclIndex on a workstation running the Tcl shell:

lib1.tcl

Example:

```
proc test1 {} {
    puts "In procedure test1"
}

proc test2 {} {
```

```
    puts "In procedure test2"
}
```

lib2.tcl

Example:

```
proc test3 {} {
    puts "In procedure test3"
}
```

Step 2 **tclsh**

Use this command to enter the Tcl shell.

Example:

```
workstation% tclsh
```

Step 3 **auto_mkindex** *directory_name* ***.tcl**

Use the **auto_mkindex** command to create the tclIndex file. The tclIndex file that contains a directory of all the procedures contained in the Tcl library files. We recommend that you run auto_mkindex inside a directory because there can only be a single tclIndex file in any directory and you may have other Tcl files to be grouped together. Running auto_mkindex in a directory determines which tcl source file or files are indexed using a specific tclIndex.

Example:

```
workstation% auto_mkindex eem_library *.tcl
```

The following example TclIndex is created when the lib1.tcl and lib2.tcl files are in a library file directory and the **auto_mkindex** command is run.

tclIndex

Example:

```
# Tcl autoload index file, version 2.0
# This file is generated by the "auto_mkindex" command
# and sourced to set up indexing information for one or
# more commands.  Typically each line is a command that
# sets an element in the auto_index array, where the
# element name is the name of a command and the value is
# a script that loads the command.

set auto_index(test1) [list source [file join $dir lib1.tcl]]
set auto_index(test2) [list source [file join $dir lib1.tcl]]
set auto_index(test3) [list source [file join $dir lib2.tcl]]
```

Step 4 Copy the Tcl library files and the tclIndex file to the directory used for storing user library files on the target device.

Step 5 Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device.

The directory for storing user-defined EEM policies can be the same directory used in Step 4. The following example user-defined EEM policy can be used to test the Tcl library support in EEM.

libtest.tcl

Example:

```
::cisco::eem::event_register_none
```

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

global auto_index auto_path

puts [array names auto_index]

if { [catch {test1} result]} {
    puts "calling test1 failed result = $result $auto_path"
}

if { [catch {test2} result]} {
    puts "calling test2 failed result = $result $auto_path"
}
if { [catch {test3} result]} {
    puts "calling test3 failed result = $result $auto_path"
}
```

Step 6 **enable**

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 7 **configure terminal**

Enables global configuration mode.

Example:

```
Device# configure terminal
```

Step 8 **event manager directory user library** *path*

Use this command to specify the EEM user library directory; this is the directory to which the files were copied.

Example:

```
Device(config)# event manager directory user library disk2:/eem_library
```

Step 9 **event manager directory user policy** *path*

Use this command to specify the EEM user policy directory; this is the directory to which the file was copied.

Example:

```
Device(config)# event manager directory user policy disk2:/eem_policies
```

Step 10 **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap]**

Use this command to register a user-defined EEM policy. In this example, the policy named libtest.tcl is registered.

Example:

```
Device(config)# event manager policy libtest.tcl
```

Step 11 **event manager run** *policy-name*

Use this command to manually run an EEM policy. In this example, the policy named libtest.tcl is run to test the Tcl support in EEM. The example output shows that the test for Tcl support in EEM was successful.

Example:

```
Device(config)# event manager run libtest.tcl
The following output is displayed:
01:24:37: %HA_EM-6-LOG: libtest.tcl: In procedure test1
01:24:37: %HA_EM-6-LOG: libtest.tcl: In procedure test2
01:24:37: %HA_EM-6-LOG: libtest.tcl: In procedure test3
```

Creating an EEM User Tcl Package Index

Perform this task to create a Tcl package index file that contains a directory of all the Tcl packages and version information contained in a library of Tcl package files. Tcl packages are supported, depending on your release, using the Tcl **package** keyword.

Tcl packages are located in either the EEM system library directory or the EEM user library directory. When a **package require** Tcl command is executed, the user library directory is searched first for a pkgIndex.tcl file. If the pkgIndex.tcl file is not found in the user directory, the system library directory is searched. In this task, a Tcl package directory--the pkgIndex.tcl file--is created in the appropriate library directory using the **pkg_mkIndex** command to contain information about all of the Tcl packages contained in the directory along with version information. If the index is not created, the Tcl packages will not be found when an EEM policy is run that contains a **package require** Tcl command.

Using the Tcl package support in EEM, users can gain access to packages such as XML_RPC for Tcl. When the Tcl package index is created, a Tcl script can easily make an XML-RPC call to an external entity.

Note Packages implemented in C programming code are not supported in EEM.

SUMMARY STEPS

1. On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl package files into the directory.
2. **tclsh**
3. **pkg_mkindex** *directory_name* ***.tcl**
4. Copy the Tcl library files and the pkgIndex file to the directory used for storing user library files on the target device.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device.
6. **enable**
7. **configure terminal**
8. **event manager directory user library** *path*
9. **event manager directory user policy** *path*
10. **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
11. **event manager run** *policy-name*

DETAILED STEPS

Step 1 On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl package files into the directory.

Step 2 **tclsh**

Use this command to enter the Tcl shell.

Example:

```
workstation% tclsh
```

Step 3 **pkg_mkindex** *directory_name* ***.tcl**

Use the **pkg_mkindex** command to create the pkgIndex file. The pkgIndex file contains a directory of all the packages contained in the Tcl library files. We recommend that you run pkg_mkindex inside a directory because there can only be a single pkgIndex file in any directory and you may have other Tcl files to be grouped together. Running pkg_mkindex in a directory determines which Tcl package file or files are indexed using a specific pkgIndex.

Example:

```
workstation% pkg_mkindex eem_library *.tcl
```

The following example pkgIndex is created when some Tcl package files are in a library file directory and the **pkg_mkindex** command is run.

pkgIndex

Example:

```
# Tcl package index file, version 1.1
# This file is generated by the "pkg_mkIndex" command
# and sourced either when an application starts up or
# by a "package unknown" script. It invokes the
# "package ifneeded" command to set up package-related
# information so that packages will be loaded automatically
# in response to "package require" commands. When this
# script is sourced, the variable $dir must contain the
# full path name of this file's directory.
package ifneeded xmlrpc 0.3 [list source [file join $dir xmlrpc.tcl]]
```

Step 4 Copy the Tcl library files and the pkgIndex file to the directory used for storing user library files on the target device.

Step 5 Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device.

The directory for storing user-defined EEM policies can be the same directory used in Step 4. The following example user-defined EEM policy can be used to test the Tcl package support in EEM.

packagetest.tcl

Example:

```
::cisco::eem::event_register_none maxrun 1000000.000
#
# test if xmlrpc available
#
#
# Namespace imports
```

```
#
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
#
package require xmlrpc
puts "Did you get an error?"
```

Step 6 **enable**

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 7 **configure terminal**

Enables global configuration mode.

Example:

```
Device# configure terminal
```

Step 8 **event manager directory user library** *path*

Use this command to specify the EEM user library directory; this is the directory to which the files in were copied.

Example:

```
Device(config)# event manager directory user library disk2:/eem_library
```

Step 9 **event manager directory user policy** *path*

Use this command to specify the EEM user policy directory; this is the directory to which the file was copied.

Example:

```
Device(config)# event manager directory user policy disk2:/eem_policies
```

Step 10 **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]

Use this command to register a user-defined EEM policy. In this example, the policy named packagetest.tcl is registered.

Example:

```
Device(config)# event manager policy packagetest.tcl
```

Step 11 **event manager run** *policy-name*

Use this command to manually run an EEM policy. In this example, the policy named packagetest.tcl is run to test the Tcl package support in EEM.

Example:

```
Device(config)# event manager run packagetest.tcl
```

Configuration Examples for Writing Embedded Event Manager Policies Using Tcl

Assigning a Username for a Tcl Session Examples

The following example shows how to set a username to be associated with a Tcl session. If you are using authentication, authorization, and accounting (AAA) security and implement authorization on a command basis, you should use the **event manager session cli username** command to set a username to be associated with a Tcl session. The username is used when a Tcl policy executes a CLI command. TACACS+ verifies each CLI command using the username associated with the Tcl session that is running the policy. Commands from Tcl policies are not usually verified because the device must be in privileged EXEC mode to register the policy. In the example, the username is yourname, and this is the username that is used whenever a CLI command session is initiated from within an EEM policy.

```
configure terminal
 event manager session cli username yourname
 end
```

EEM Event Detector Demo Examples

EEM Sample Policy Descriptions

This configuration example features some of the sample EEM policies:

- ap_perf_test_base_cpu.tcl--Is run to measure the the CPU performance of EEM policies.
- no_perf_test_init.tcl--Is run to measure the CPU performance of EEM policies.
- sl_intf_down.tcl--Is run when a configurable syslog message is logged. It executes up to two configurable CLI commands and e-mails the results.
- tm_cli_cmd.tcl--Is run using a configurable CRON entry. It executes a configurable CLI command and e-mails the results.
- tm_crash_reporter.tcl--Is run 5 seconds after it is registered and 5 seconds after the device boots up. When triggered, the script attempts to find the reload reason. If the reload reason was due to a crash, the policy searches for the related crashinfo file and sends this information to a URL location specified by the user in the environment variable _crash_reporter_url.
- tm_fsys_usage.tcl--This policy runs using a configurable CRON entry and monitors disk space usage. A syslog message is displayed if disk space usage crosses configurable thresholds.

Event Manager Environment Variables for the Sample Policies

Event manager environment variables are Tcl global variables that are defined external to the EEM policy before the policy is registered and run. The sample policies require three of the e-mail environment variables to be set ; only _email_cc is optional. Other required and optional variable settings are outlined in the following tables.

The table below describes the EEM environment variables that must be set before the ap_perf_test_base_cpu.tcl sample policy is run.

Table 9: Environment Variables Used in the ap_perf_test_base_cpu.tcl Policy

Environment Variable	Description	Example
_perf_iterations	The number of iterations over which to run the measurement.	**100**
_perf_cmd1	The first non interactive CLI command that is executed as part of the measurement test. This variable is optional and need not be specified.	**enable**
_perf_cmd2	The second non interactive CLI command that is as part of the measurement test. To use _perf_cmd2, _perf_cmd1 must be defined. This variable is optional and need not be specified.	**show version**
_perf_cmd3	The third non interactive CLI command that is as part of the measurement test. To use _perf_cmd3, _perf_cmd1 must be defined. This variable is optional and need not be specified.	**show interface counters protocol status**

The table below describes the EEM environment variables that must be set before the no_perf_test_init.tcl sample policy is run.

Table 10: Environment Variables Used in the no_perf_test_init.tcl Policy

Environment Variable	Description	Example
_perf_iterations	The number of iterations over which to run the measurement.	**100**
_perf_cmd1	The first non interactive CLI command that is executed as part of the measurement test. This variable is optional and need not be specified.	**enable**
_perf_cmd2	The second non interactive CLI command that is as part of the measurement test. To use _perf_cmd2, _perf_cmd1 must be defined. This variable is optional and need not be specified.	**show version**
_perf_cmd3	The third non interactive CLI command that is as part of the measurement test. To use _perf_cmd3, _perf_cmd1 must be defined. This variable is optional and need not be specified.	**show interface counters protocol status**

The table below describes the EEM environment variables that must be set before the sl_intf_down.tcl sample policy is run.

Table 11: Environment Variables Used in the sl_intf_down.tcl Policy

Environment Variable	Description	Example
_config_cmd1	The first configuration command that is executed.	**interface Ethernet1/0**

Environment Variable	Description	Example
_config_cmd2	The second configuration command that is executed. This variable is optional and need not be specified.	**no shutdown**
_syslog_pattern	A regular expression pattern match string that is used to compare syslog messages to determine when the policy runs.	.*UPDOWN.*FastEthernet0/0.*

The table below describes the EEM environment variables that must be set before the tm_cli_cmd.tcl sample policy is run.

Table 12: Environment Variables Used in the tm_cli_cmd.tcl Policy

Environment Variable	Description	Example
_cron_entry	A CRON specification that determines when the policy will run.	0-59/1 0-23/1 * * 0-7
_show_cmd	The CLI command to be executed when the policy is run.	**show version**

The table below describes the EEM environment variables that must be set before the tm_crash_reporter.tcl sample policy is run.

Table 13: Environment Variables Used in the tm_crash_reporter.tcl Policy

Environment Variable	Description	Example
_crash_reporter_debug	A value that identifies whether debug information for tm_crash_reporter.tcl will be enabled. This variable is optional and need not be specified.	1
_crash_reporter_url	The URL location to which the crash report is sent.	http://www.example.com/fm/interface_tm.cgi

The table below describes the EEM environment variables that must be set before the tm_fsys_usage.tcl sample policy is run.

Table 14: Environment Variables Used in the tm_fsys_usage.tcl Policy

Environment Variable	Description	Example
_tm_fsys_usage_cron	A CRON specification that is used in the **event_register** Tcl command extension. If unspecified, the tm_fsys_usage.tcl policy is triggered once per minute. This variable is optional and need not be specified.	0-59/1 0-23/1 * * 0-7
_tm_fsys_usage_debug	When this variable is set to a value of 1, disk usage information is displayed for all entries in the system. This variable is optional and need not be specified.	1

Environment Variable	Description	Example
_tm_fsys_usage_ freebytes	Free byte threshold for systems or specific prefixes. If free space falls below a given value, a warning is displayed. This variable is optional and need not be specified.	disk2:98000000
_tm_fsys_usage_percent	Disk usage percentage thresholds for systems or specific prefixes. If the disk usage percentage exceeds a given percentage, a warning is displayed. If unspecified, the default disk usage percentage is 80 percent for all systems. This variable is optional and need not be specified.	nvram:25 disk2:5

Registration of Some EEM Policies

Some EEM policies must be unregistered and then reregistered if an EEM environment variable is modified after the policy is registered. The event_register_*xxx* statement that appears at the start of the policy contains some of the EEM environment variables, and this statement is used to establish the conditions under which the policy is run. If the environment variables are modified after the policy has been registered, the conditions may become invalid. To avoid any errors, the policy must be unregistered and then reregistered. The following variables are affected:

- _cron_entry in the tm_cli_cmd.tcl policy
- _syslog_pattern in the sl_intf_down.tcl policy

Basic Configuration Details for All Sample Policies

To allow e-mail to be sent from the Embedded Event Manager, the **hostname** and **ip domain-name** commands must be configured. The EEM environment variables must also be set. After a Cisco IOS image has been booted, use the following initial configuration, substituting appropriate values for your network. The environment variables for the tm_fsys_usage sample policy (see the table above) are all optional and are not listed here:

```
hostname cpu
ip domain-name example.com
event manager environment _email_server ms.example.net
event manager environment _email_to username@example.net
event manager environment _email_from engineer@example.net
event manager environment _email_cc projectgroup@example.net
event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7
event manager environment _show_cmd show event manager policy registered
event manager environment _syslog_pattern .*UPDOWN.*FastEthernet0/0
event manager environment _config_cmd1 interface Ethernet1/0
event manager environment _config_cmd2 no shutdown
event manager environment _crash_reporter_debug 1
event manager environment _crash_reporter_url
http://www.example.com/fm/interface_tm.cgi
end
```

Using the Sample Policies

This section contains the following configuration scenarios to demonstrate how to use the some sample Tcl policies:

Running the Mandatory.go_*.tcl Sample Policy

There are GOLD TCL scripts for each test which runs as a part of GOLD EEM Policy. You can modify the TCL script for the test, specify the consecutive failure count, and also change the default corrective action. For example, one could chose to power down a linecard card, instead of reset or other CLI based actions.

For each registered test, a default TCL script is available, which can be registered with the system, and matches with the default action. This can be then overridden by modifying these scripts.

The following table shows a list of the mandatory polices that GOLD installed into EEM. Each of the policies performs some sort of action such as resetting the card or disabling the port.

GOLD Tcl Scripts	Test
Mandatory.go_asicsync.tcl	TestAsicSync
Mandatory.go_bootup.tcl	Common for all bootup tests.
Mandatory.go_fabric.tcl	TestFabricHealth
Mandatory.go_fabrich0.tcl	TestFabricCh0Health
Mandatory.go_fabrich1.tcl	TestFabricCh1Health
Mandatory.go_ipsec.tcl	TestIPSecEncrypDecrypPkt
Mandatory.go_mac.tcl	TestMacNotification
Mandatory.go_nondislp.tcl	TestNonDisruptiveLoopback
Mandatory.go_scratchreg.tcl	TestScratchRegister
Mandatory.go_sprping.tcl	TestSPRPInbandPing

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode, where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the mandatory.go_*.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command again to verify that the policy has been registered.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy Mandatory.go_spuriousisr.tcl
 end
show event manager policy registered
show event manager environment
```

Running the ap_perf_test_base_cpu.tcl and no_perf_test_init.tcl Sample Policies

These sample policies measures the CPU performance of EEM policies. The policies help find the average execution time of each EEM policy and uses the CLI library to execute the configuration commands specified in the EEM environment variables _perf_cmd1 and, optionally, _perf_cmd2 and _perf_cmd3.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode, where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, enter the **service timestamps debug datetime msec** command and then you can register the ap_perf_test_base_cpu.tcl and no_perf_test_init.tcl policies with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command again to verify that the policy has been registered.

The policies ap_perf_test_base_cpu.tcl and no_perf_test_init.tcl need to be registered together, as they run as a test suite. You can run the no_perf_test_init.tcl policy to start the tests. Analyze the results using the syslog messages from each iteration. The total number of iteration is specified by the variable _perf_iterations. Take the time difference and divide it by the total number of iterations to get the average execution time of each EEM policy.

```
enable
show event manager policy registered
show event manager policy available
show event manager environment
configure terminal
 service timestamps debug datetime msec
 event manager environment _perf_iterations 100
 event manager policy ap_perf_test_base_cpu.tcl
 event manager policy no_perf_test_init.tcl
 end
show event manager policy registered
show event manager policy available
show event manager environment
event manager run no_perf_test_init.tcl
```

Running the no_perf_test_init.tcl Sample Policy

This sample policy measures the the cpu performance of EEM policies. The policy helps to find the average execution time of each EEM policy and uses the CLI library to execute the configuration commands specified in the EEM environment variables _perf_cmd1 and, optionally, _perf_cmd2 and _perf_cmd3.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode, where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the no_perf_test_init.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command again to verify that the policy has been registered.

Analyze the results using the syslog messages from each iteration. The total number of iteration is specified by the variable _perf_iterations. Take the time difference and divide it by the total number of iterations to get the average execution time of each EEM policy.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy no_perf_test_init.tcl
 end
```

```
show event manager policy registered
show event manager environment
```

Running the sl_intf_down.tcl Sample Policy

This sample policy demonstrates the ability to modify the configuration when a syslog message with a specific pattern is logged. The policy gathers detailed information about the event and uses the CLI library to execute the configuration commands specified in the EEM environment variables _config_cmd1 and, optionally, _config_cmd2. An e-mail message is sent with the results of the CLI command.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode, where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the sl_intf_down.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command again to verify that the policy has been registered.

The policy runs when an interface goes down. Enter the **show event manager environment** command to display the current environment variable values. Unplug the cable (or configure a shutdown) for the interface specified in the _syslog_pattern EEM environment variable. The interface goes down, prompting the syslog daemon to log a syslog message about the interface being down, and the syslog event detector is called.

The syslog event detector reviews the outstanding event specifications and finds a match for interface status change. The EEM server is notified, and the server runs the policy that is registered to handle this event--sl_intf_down.tcl.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy sl_intf_down.tcl
 end
show event manager policy registered
show event manager environment
```

Running the tm_cli_cmd.tcl Sample Policy

This sample policy demonstrates the ability to periodically execute a CLI command and to e-mail the results. The CRON specification “0-59/2 0-23/1 * * 0-7” causes this policy to be run on the second minute of each hour. The policy gathers detailed information about the event and uses the CLI library to execute the configuration commands specified in the EEM environment variable _show_cmd. An e-mail message is sent with the results of the CLI command.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the tm_cli_cmd.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command to verify that the policy has been registered.

The timer event detector triggers an event for this case periodically according to the CRON string set in the EEM environment variable _cron_entry. The EEM server is notified, and the server runs the policy that is registered to handle this event--tm_cli_cmd.tcl.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_cli_cmd.tcl
 end
show event manager policy registered
```

Running the tm_crash_reporter.tcl Sample Policy

This sample policy demonstrates the ability to send an HTTP-formatted crash report to a URL location. If the policy registration is saved in the startup configuration file, the policy is triggered 5 seconds after bootup. When triggered, the script attempts to find the reload reason. If the reload reason was due to a crash, the policy searches for the related crashinfo file and sends this information to a URL location specified by the user in the environment variable _crash_reporter_url. A CGI script, interface_tm.cgi, has been created to receive the URL from the tm_crash_reporter.tcl policy and save the crash information in a local database on the target URL machine.

A Perl CGI script, interface_tm.cgi, has been created and is designed to run on a machine that contains an HTTP server and is accessible by the device that runs the tm_crash_reporter.tcl policy. The interface_tm.cgi script parses the data passed into it from tm_crash_reporter.tcl and appends the crash information to a text file, creating a history of all crashes in the system. Additionally, detailed information on each crash is stored in three files in a crash database directory that is specified by the user. Another Perl CGI script, crash_report_display.cgi, has been created to display the information stored in the database created by the interface_tm.cgi script. The crash_report_display.cgi script should be placed on the same machine that contains interface_tm.cgi. The machine should be running a web browser such as Internet Explorer or Netscape. When the crash_report_display.cgi script is run, it displays the crash information in a readable format.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the tm_crash_reporter.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command to verify that the policy has been registered.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_crash_reporter.tcl
 end
show event manager policy registered
```

Running the tm_fsys_usage.tcl Sample Policy

This sample policy demonstrates the ability to periodically monitor disk space usage and report through syslog when configurable thresholds have been crossed.

The following sample configuration demonstrates how to use this policy. Starting in user EXEC mode, enter the **enable** command at the device prompt. The device enters privileged EXEC mode, where you can enter the **show event manager policy registered** command to verify that no policies are currently registered. The next command is the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure terminal** command to reach global configuration mode, you can register the tm_fsys_usage.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command again to verify that the policy has been registered. If you had configured any of the optional environment variables that are used in the tm_fsys_usage.tcl policy, the **show event manager environment** command displays the configured variables.

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_fsys_usage.tcl
 end
show event manager policy registered
show event manager environment
```

Programming Policies with Tcl Sample Scripts Example

This section contains some of the sample policies that are included as EEM system policies. For more details about these policies, see the EEM Event Detector Demo Examples, on page 37.

Mandatory.go_ipsec.tcl Sample Policy

The following sample policy for the TestIPSecEncrypDecrypPkt Test.

```
::cisco::eem::event_register_gold card all testing_type monitoring test_name Tes
tIPSecEncrypDecrypPkt consecutive_failure 6 platform_action 0 queue_priority las
t
#
#  GOLD TestIPSecEncrypDecrypPkt Test TCL script
#
#  March 2005, Hai Qiu
#
#  Copyright (c) 2005-2007 by cisco Systems, Inc.
#  All rights reserved.
#
#
# Register for TestIPSecEncrypDecrypPkt test even
# the elements for register the event
# card [all | card #]
# sub_card [all | sub_card #]
# severity_major | severity_minor | severity_normal  default :  severity_normal
# new_failure [true | false]  default: dont_care
# testing_type [bootup | ondemand | schedule | monitoring]
# test_name [ test name ]
# test_id [ test # ]
# consecutive_failure [ consecutive_failure # ]
# platform_action [action_flag]
# action_flag [ 0 | 1 | 2 ]
# queue_priority [ normal | low | high | last]   default: normal
#
# Note:
#    1: "card" element is required. If other elements are not specified,
#       treat them as dont care, or default.
```

```
#
#    2: action_flag is platform specific. It is up to platform to
#       determine what action need to be taken based on the value
#       For Cat6k platform
#       action_flag 0 : TCL script take action to reset card
#       action_flag 1 : TCL script doesn't take action to reset card
#       action_flag 2 : TCL script takes action to reset card for bootup diag
#                       when there is major error
#       action_flag 3 : TCL script doesn't take action to reset card for
#                       bootup diag when there is major error
#
#    3: "queue_priority last" would guarantee this policy will be executed last
#       if there are other EEM events in queue with queue priority other
#       than "last"
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# 1. query the information of latest triggered eem event
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
puts "GOLD EEM TCL policy for TestIPSecEncrypDecrypPkt"
#set msg [format "array=%s", array names arr_einfo]
#puts "msg $msg"
#set msg $arr_einfo(msg)
set card $arr_einfo(card)
set sub_card $arr_einfo(sub_card)
#set overall_result $arr_einfo(overall_result)
#puts "GOLD event msg recieved: $card/$sub_card overall_result= $overall_result"
# 2. execute the user-defined config commands
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "en"} result] {
    error $result $errorInfo
}
# Use "diagn action mod mod# test testname default" command
# for default platform action
if [catch {cli_exec $cli1(fd) "diagnostic action mod $card test TestIPSecEncrypD
ecrypPkt default"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}
```

ap_perf_test_base_cpu.tcl Sample Policy

The following sample policy measures the CPU performance of EEM policies.

```
::cisco::eem::event_register_appl sub_system 798 type 9999
#----------------------------------
# EEM policy used for measuring the cpu performance of EEM policies.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005, 2006 by cisco Systems, Inc.
```

```
# All rights reserved.
#------------------
###
### Input arguments:
###
### arg1 $iter                    - current iteration count
###
### The following EEM environment variables are used:
###
### _perf_iterations (mandatory) - number of iterations over which we
###                                will run our measurement.
### Example:
### event manager environment _perf_iterations 100
###
### _perf_cmd1 (optional)        - optional non interactive cli command
###                                to be executed as part of the
###                                measurement test.
### Example:
### event manager environment _perf_cmd1 enable
###
### _perf_cmd2 (optional)        - optional non interactive cli command
###                to be executed as part of the
###                                measurement test.
###                                To use _perf_cmd2, _perf_cmd1 MUST
###                be defined.
### Example:
### event manager environment _perf_cmd2 show ver
###
### _perf_cmd3 (optional)        - optional non interactive cli command
###                                to be executed as part of the
###                                measurement test.
###                                To use _perf_cmd3, _perf_cmd1 MUST
###                                be defined.
### Example:
### event manager environment _perf_cmd3 show int counters protocol status
###
### Description:
###   Iterate through _perf_iterations of this policy.
###   It is up to the user to calculate the average
###   execution time based on the system timestamps.
###   Optional commands _perf_cmd1,
###   _perf_cmd2 and _perf_cmd3 are executed if defined.
###
###   A value of 100 is a good starting point.
###
### Outputs:
###   Console output.
###
### Usage example:
###   >conf t
###   >service timestamps debug datetime msec
###   >event manager environment _perf_iterations 100
###   >event manager policy ap_perf_base_cpu.tcl
###   >event manager policy no_perf_test_init.tcl
###   >end
###   2d19h: %SYS-5-CONFIG_I: Configured from console by console
###   >event manager run no_perf_test_init.tcl
###
###   Oct 16 14:57:17.284: %SYS-5-CONFIG_I: Configured from console by console
###   >event manager run no_perf_test_init.tcl
###
###   Oct 16 19:32:02.772: %HA_EM-6-LOG:
###     eem_policy/no_perf_test_init.tcl: EEM performance test start
###   Oct 16 19:32:03.115: %HA_EM-6-LOG:
```

```
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 1
###   Oct 16 19:32:03.467: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 2
###   ...
###   Oct 16 19:32:36.936: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 100
###   Oct 16 19:32:36.936: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test end
###
###   The user must calculate execution time and average time of execution.
###   In this example, total time = 19:32:36.936 - 19:32:02.772 = 34.164
###   Average script execution time = 341.64 milliseconds
###
# check if all the env variables we need exist
# If any of them doesn't exist, print out an error msg and quit
if {![info exists _perf_iterations]} {
    set result \
        "Policy cannot be run: variable _perf_iterations has not been set"
    error $result $errorInfo
}
# ensure our target iteration count > 0
if {$_perf_iterations <= 0} {
    set result \
        "Policy cannot be run: variable _perf_iterations <= 0"
    error $result $errorInfo
}
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# query the event info
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
set iter $arr_einfo(data1)
set iter [expr $iter + 1]
# if _perf_cmd1 is defined
if {[info exists _perf_cmd1]} {
    # open the cli library
    if [catch {cli_open} result] {
        error $result $errorInfo
    } else {
        array set cli1 $result
    }
    # execute the comamnd defined in _perf_cmd1
    if [catch {cli_exec $cli1(fd) $_perf_cmd1} result] {
        error $result $errorInfo
    }
    # if _perf_cmd2 is defined
    if {[info exists _perf_cmd2]} {
        # execute the comamnd defined in _perf_cmd2
        if [catch {cli_exec $cli1(fd) $_perf_cmd2} result] {
            error $result $errorInfo
        } else {
            set cmd_output $result
        }
    }
    # if _perf_cmd3 is defined
    if {[info exists _perf_cmd3]} {
        # execute the comamnd defined in _perf_cmd3
        if [catch {cli_exec $cli1(fd) $_perf_cmd3} result] {
            error $result $errorInfo
        } else {
```

```
            set cmd_output $result
        }
    }
    # close the cli library
    if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
        error $result $errorInfo
    }
}

# log a message
set msg [format "EEM performance test iteration %s" $iter]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# use the context info from the previous run to determine when to end
if {$iter >= $_perf_iterations} {
    #log the final messages
    action_syslog priority info msg "EEM performance test end"
    if {$_cerrno != 0} {
        set result [format \
                "component=%s; subsys err=%s; posix err=%s;\n%s" \
                $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
        error $result
    }
    exit 0
}
# cause the next iteration to run
event_publish sub_system 798 type 9999 arg1 $iter
if {$_cerrno != 0} {
    set result [format \
            "component=%s; subsys err=%s; posix err=%s;\n%s" \
            $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

tm_cli_cmd.tcl Sample Policy

The following sample policy runs a configurable CRON entry. The policy executes a configurable Cisco IOS CLI command and e-mails the results. An optional log file can be defined to which the output is appended with a timestamp.

```
::cisco::eem::event_register_timer cron name crontimer2 cron_entry $
_cron_entry maxrun 240
#-------------------------------------------------------------------
# EEM policy that will periodically execute a cli command and email the
# results to a user.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------
### The following EEM environment variables are used:
###
### _cron_entry (mandatory)            - A CRON specification that determines
###                                      when the policy will run. See the
###                                      IOS Embedded Event Manager
###                                      documentation for more information
###                                      on how to specify a cron entry.
```

```
### Example: _cron_entry                0-59/1 0-23/1 * * 0-7
###
### _log_file (mandatory without _email_....)
###                                     - A filename to append the output to.
###                                       If this variable is defined, the
###                                       output is appended to the specified
###                                       file with a timestamp added.
### Example: _log_file                  bootflash:/my_file.log
###
### _email_server (mandatory without _log_file)
###                                     - A Simple Mail Transfer Protocol (SMTP)
###                                       mail server used to send e-mail.
### Example: _email_server              mailserver.example.com
###
### _email_from (mandatory without _log_file)
###                                     - The address from which e-mail is sent.
### Example: _email_from                devtest@example.com
###
### _email_to (mandatory without _log_file)
###                                     - The address to which e-mail is sent.
### Example: _email_to                  engineering@example.com
###
### _email_cc (optional)                - The address to which the e-mail must
###                                       be copied.
### Example: _email_cc                  manager@example.com
###
### _show_cmd (mandatory)               - The CLI command to be executed when
###                                       the policy is run.
### Example: _show_cmd                  show version
###
# check if all required environment variables exist
# If any required environment variable does not exist, print out an error msg and quit
if {![info exists _log_file]} {
    if {![info exists _email_server]} {
 set result \
  "Policy cannot be run: variable _log_file or _email_server has not been set"
 error $result $errorInfo
    }
    if {![info exists _email_from]} {
 set result \
  "Policy cannot be run: variable _log_file or _email_from has not been set"
 error $result $errorInfo
    }
    if {![info exists _email_to]} {
 set result \
  "Policy cannot be run: variable _log_file ore _email_to has not been set"
 error $result $errorInfo
    }
    if {![info exists _email_cc]} {
 #_email_cc is an option, must set to empty string if not set.
 set _email_cc ""
    }
}
if {![info exists _show_cmd]} {
    set result \
        "Policy cannot be run: variable _show_cmd has not been set"
    error $result $errorInfo
}
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# query the event info and log a message
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
```

```
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
global timer_type timer_time_sec
set timer_type $arr_einfo(timer_type)
set timer_time_sec $arr_einfo(timer_time_sec)
# log a message
set msg [format "timer event: timer type %s, time expired %s" \
        $timer_type [clock format $timer_time_sec]]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
 $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# 1. execute the command
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "en"} result] {
    error $result $errorInfo
}
# save exact execution time for command
set time_now [clock seconds]
# execute command
if [catch {cli_exec $cli1(fd) $_show_cmd} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
    # format output: remove trailing router prompt
    regexp {\n*(.*\n)([^\n]*)$} $result dummy cmd_output
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}

# 2. log the success of the CLI command
set msg [format "Command \"%s\" executed successfully" $_show_cmd]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# 3. if _log_file is defined, then attach it to the file
if {[info exists _log_file]} {
    # attach output to file
    if [catch {open $_log_file a+} result] {
        error $result
    }
    set fileD $result
    # save timestamp of command execution
    #      (Format = 00:53:44 PDT Mon May 02 2005)
    set time_now [clock format $time_now -format "%T %Z %a %b %d %Y"]
    puts $fileD "%%% Timestamp = $time_now"
    puts $fileD $cmd_output
    close $fileD
}
# 4. if _email_server is defined send the email out
if {[info exists _email_server]} {
    set routername [info hostname]
    if {[string match "" $routername]} {
```

```
 error "Host name is not configured"
    }
    if [catch {smtp_subst [file join $tcl_library email_template_cmd.tm]} \
     result] {
 error $result $errorInfo
    }
    if [catch {smtp_send_email $result} result] {
 error $result $errorInfo
    }
}
```

sl_intf_down.tcl Sample Policy

The following sample policy runs when a configurable syslog message is logged. The policy executes a configurable CLI command and e-mails the results.

```
::cisco::eem::event_register_syslog occurs 1 pattern $_syslog_pattern maxrun 90

#-------------------------------------------------------------------
# EEM policy to monitor for a specified syslog message.
# Designed to be used for syslog interface-down messages.
# When event is triggered, the given config commands will be run.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------

### The following EEM environment variables are used:
###
### _syslog_pattern (mandatory)        - A regular expression pattern match string
###                                      that is used to compare syslog messages
###                                      to determine when policy runs
### Example: _syslog_pattern             .*UPDOWN.*FastEthernet0/0.*
###
### _email_server (mandatory)          - A Simple Mail Transfer Protocol (SMTP)
###                                      mail server used to send e-mail.
### Example: _email_server               mailserver.example.com
###
### _email_from (mandatory)            - The address from which e-mail is sent.
### Example: _email_from                 devtest@example.com
###
### _email_to (mandatory)              - The address to which e-mail is sent.
### Example: _email_to                   engineering@example.com
###
### _email_cc (optional)               - The address to which the e-mail must
###                                      be copied.
### Example: _email_cc                   manager@example.com
###
### _config_cmd1 (optional)            - The first configuration command that
###                                      is executed.
### Example: _config_cmd1                interface Ethernet1/0
###
### _config_cmd2 (optional)            - The second configuration command that
###                                      is executed.
### Example: _config_cmd2                no shutdown
###

# check if all the env variables we need exist
# If any of them doesn't exist, print out an error msg and quit
if {![info exists _email_server]} {
    set result \
```

```
        "Policy cannot be run: variable _email_server has not been set"
    error $result $errorInfo
}
if {![info exists _email_from]} {
    set result \
        "Policy cannot be run: variable _email_from has not been set"
    error $result $errorInfo
}
if {![info exists _email_to]} {
    set result \
        "Policy cannot be run: variable _email_to has not been set"
    error $result $errorInfo
}
if {![info exists _email_cc]} {
     #_email_cc is an option, must set to empty string if not set.
     set _email_cc ""
}

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

# 1. query the information of latest triggered eem event
array set arr_einfo [event_reqinfo]

if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
      $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

set msg $arr_einfo(msg)
set config_cmds ""

# 2. execute the user-defined config commands
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "en"} result] {
    error $result $errorInfo
}
if [catch {cli_exec $cli1(fd) "config t"} result] {
    error $result $errorInfo
}

if {[info exists _config_cmd1]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd1} result] {
        error $result $errorInfo
    }
    append config_cmds $_config_cmd1
}

if {[info exists _config_cmd2]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd2} result] {
        error $result $errorInfo
    }
    append config_cmds "\n"
    append config_cmds $_config_cmd2
}

if [catch {cli_exec $cli1(fd) "end"} result] {
    error $result $errorInfo
}
```

```
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}

after 60000
# 3. send the notification email
set routername [info hostname]
if {[string match "" $routername]} {
    error "Host name is not configured"
}

if [catch {smtp_subst [file join $tcl_library email_template_cfg.tm]} result] {
    error $result $errorInfo
}
if [catch {smtp_send_email $result} result] {
    error $result $errorInfo
}
```

The following e-mail template file is used with the EEM sample policy above:

```
email_template_cfg.tm
Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc: $_email_cc
Subject: From router $routername: Periodic $_show_cmd Output
$cmd_output
```

Debugging Embedded Event Manager Policies Examples

The following examples show how to debug the CLI library and the SMTP library.

Debugging the CLI Library

The CLI library allows users to run CLI commands and obtain the output of commands in Tcl. An Embedded Event Manager **debug** command has been provided for users of this library. The command to enable CLI library debugging is **debug event manager tcl cli_library**. When enabled, this command displays all data that is passed in and read back from the TTY session that handles the CLI interactions. This data helps ensure users that the commands that they are passing to the CLI are valid.

Example of the debug event manager tcl cli_library Command

This example uses the sample policy sl_intf_down.tcl. When triggered, sl_intf_down.tcl passes a configuration command to the CLI through the CLI library. The command passed in below is **show event manager environment**. This command is not a valid command in configuration mode. Without the **debug** command enabled, the output is shown below:

```
00:00:57:sl_intf_down.tcl[0]:config_cmds are show eve man env
00:00:57:%SYS-5-CONFIG_I:Configured from console by vty0
```

Notice that with the output above the user would not know whether or not the command succeeded in the CLI. With the **debug event manager tcl cli_library** command enabled, the user sees the following:

```
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : CTL : cli_open called.
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : nelson>
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : IN  : nelson>enable
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : nelson#
```

```
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : IN  : nelson#configure terminal
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : Enter configuration commands, one
per line.  End with CNTL/Z.
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : nelson(config)#
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : IN  : nelson(config)#show event manager
environment
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT :                 ^
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : % Invalid input detected at '^'
marker.
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : nelson(config)#
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : IN  : nelson(config)#end
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : OUT : nelson#
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : CTL : cli_close called.
01:17:07: sl_intf_down.tcl[0]: DEBUG(cli_lib) : IN  : nelson#exit
01:17:07: sl_intf_down.tcl[0]: config_cmds are show event manager environment
01:17:07: %SYS-5-CONFIG_I: Configured from console by vty0
```

The output above shows that **show event manager environment** is an invalid command in configuration mode. The IN keyword signifies all data passed in to the TTY through the CLI library. The OUT keyword signifies all data read back from the TTY through the CLI library. The CTL keyword signifies helper functions used in the CLI library. These helper functions are used to set up and remove connections to the CLI.

Debugging the SMTP Library

The SMTP library allows users to send e-mail messages to an SMTP e-mail server. An Embedded Event Manager **debug** command has been provided for users of this library. The command to enable SMTP library debugging is **debug event manager tcl smtp_library**. When enabled, this command displays all data that is passed in and read back from the SMTP library routines. This data helps ensure users that the commands that they are passing to the SMTP library are valid.

Example of the debug event manager tcl smtp_library Command

This example uses the sample policy tm_cli_cmd.tcl. When triggered, tm_cli_cmd.tcl runs the command **show event manager policy available system** through the CLI library. The result is then mailed to a user through the SMTP library. The output will help debug any issues related to using the SMTP library.

With the **debug event manager tcl smtp_library** command enabled, the users see the following on the console:

```
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 220 XXXX.example.com ESMTP XXXX
 1.1.0; Tue,
25 Jun 2002 14:20:39 -0700 (PDT)
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : HELO XXXX.example.com
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 XXXX.example.com Hello
XXXX.example.com [XXXX],
pleased to meet you
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : MAIL FROM:<XX@example.com>
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Sender
 ok
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : RCPT TO:<XX@example.com>
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Recipient
 ok
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : RCPT TO:<XX@example.com>
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Recipient
 ok
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : DATA
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 354 Enter mail, end with "."
on a line by itself
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : Date: 25 Jun 2002 14:35:00 UTC
```

```
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : Message-ID:
<20020625143500.2387058729877@XXXX.example.com>
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : From: XX@example.com
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : To: XX@example.com
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : Cc: XX@example.com
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : Subject: From router nelson:
Periodic show eve man po ava system Output
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : No. Type    Time Created
        Name
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : 1   system  Fri May3   20:42:34
 2002  pr_cdp_abort.tcl
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : 2   system  Fri May3   20:42:54
 2002  pr_iprouting_abort.tcl
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : 3   system  Wed Apr3   02:16:33
 2002  sl_intf_down.tcl
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : 4   system  Mon Jun24  23:34:16
 2002  tm_cli_cmd.tcl
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : 5   system  Wed Mar27  05:53:15
 2002  tm_crash_hist.tcl
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : nelson#
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write :
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : .
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 ADE90179 Message accepted
for delivery
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : QUIT
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 221 XXXX.example.com closing
connection
```

Tracing Tcl set Command Operations Example

Tcl is a flexible language. One of the flexible aspects of Tcl is that you can override commands. In this example, the Tcl **set** command is renamed as _set and a new version of the **set** command is created that displays a message containing the text “setting” and appends the scalar variable that is being set. This example can be used to trace all instances of scalar variables being set.

```
rename set _set
proc set {var args} {
   puts [list setting $var $args]
   uplevel _set $var $args
};
```

When this is placed in a policy, a message is displayed anytime a scalar variable is set, for example:

```
02:17:58: sl_intf_down.tcl[0]: setting test_var 1
```

RPC Event Detector Example

```
TCL script (rpccli.tcl):
::cisco::eem::event_register_rpc
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
proc run_cli { clist } {
    set rbuf ""
    if {[llength $clist] < 1} {
    return -code ok $rbuf
    }
    if {[catch {cli_open} result]} {
        return -code error $result
```

```
    } else {
    array set cliarr $result
    }
    if {[catch {cli_exec $cliarr(fd) "enable"} result]} {
        return -code error $result
    }
    if {[catch {cli_exec $cliarr(fd) "term length 0"} result]} {
        return -code error $result
    }
    foreach cmd $clist {
    if {[catch {cli_exec $cliarr(fd) $cmd} result]} {
            return -code error $result
    }
    append rbuf $result
    }
    if {[catch {cli_close $cliarr(fd) $cliarr(tty_id)} result]} {
        puts "WARNING: $result"
    }
    return -code ok $rbuf
}
proc run_cli_interactive { clist } {
    set rbuf ""
    if {[llength $clist] < 1} {
    return -code ok $rbuf
    }
    if {[catch {cli_open} result]} {
        return -code error $result
    } else {
    array set cliarr $result
    }
    if {[catch {cli_exec $cliarr(fd) "enable"} result]} {
        return -code error $result
    }
    if {[catch {cli_exec $cliarr(fd) "term length 0"} result]} {
        return -code error $result
    }
    foreach cmd $clist {
        array set sendexp $cmd
    if {[catch {cli_write $cliarr(fd) $sendexp(send)} result]} {
            return -code error $result
    }
    foreach response $sendexp(responses) {
        array set resp $response
        if {[catch {cli_read_pattern $cliarr(fd) $resp(expect)} result]} {
                return -code error $result
        }
        if {[catch {cli_write $cliarr(fd) $resp(reply)} result]} {
                return -code error $result
        }
    }
    if {[catch {cli_read $cliarr(fd)} result]} {
            return -code error $result
    }
    append rbuf $result
    }
    if {[catch {cli_close $cliarr(fd) $cliarr(tty_id)} result]} {
        puts "WARNING: $result"
    }
    return -code ok $rbuf
}
array set arr_einfo [event_reqinfo]
set args $arr_einfo(argc)
set cmds [list]
for { set i 0 } { $i < $args } { incr i } {
```

```
    set arg "arg${i}"
    # Split each argument on the '^' character.  The first element is
    # the command, and each subsequent element is a prompt followed by
    # a response to that prompt.
    set cmdlist [split $arr_einfo($arg) "^"]
    set cmdarr(send) [lindex $cmdlist 0]
    set cmdarr(responses) [list]
    if { [expr ([llength $cmdlist] - 1) % 2] != 0 } {
    return -code 88
    }
    set cmdarr(responses) [list]
    for { set j 1 } { $j < [llength $cmdlist] } { incr j 2 } {
    set resps(expect) [lindex $cmdlist $j]
    set resps(reply) [lindex $cmdlist [expr $j + 1]]
    lappend cmdarr(responses) [array get resps]
    }
    lappend cmds [array get cmdarr]
}
set rc [catch {run_cli_interactive $cmds} output]
if { $rc != 0 } {
    error $output $errorInfo
    return -code 88
}
puts $output
```

Additional References

The following sections provide references related to writing Embedded Event Manager policies using Tcl.

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
EEM commands: complete command syntax, defaults, command mode, command history, usage guidelines, and examples	Cisco IOS Embedded Event Manager Command Reference
Embedded Event Manager overview	Embedded Event Manager Overview module.
Embedded Event Manager policy writing using the CLI	Writing Embedded Event Manager Policies Using the Cisco IOS CLI module
Embedded Resource Manager	Embedded Resource Manager module

MIBs

MIB	MIBs Link
CISCO-EMBEDDED-EVENT-MGR-MIB	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
No new or modified RFCs are supported by this feature, and support for existing RFCs has not been modified by this feature.	--

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

Feature Information for Writing EEM 4.0 Policies Using the Cisco IOS CLI

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 15: Feature Information for Writing EEM 4.0 Policies Using the Cisco IOS CLI

Feature Name	Releases	Feature Information
Embedded Event Manager 4.0	15.2(5)E1	This feature was introduced and is supported only on c2960cx platform.