Configuring and Managing Embedded Event Manager Policies

The Cisco IOS XR Software Embedded Event Manager (EEM) functions as the central clearing house for the events detected by any portion of the Cisco IOS XR Software processor failover services. The EEM is responsible for detection of fault events, fault recovery, and process reliability statistics in a Cisco IOS XR Software system. The EEM events are notifications that something significant has occurred within the system, such as:

- Operating or performance statistics outside the allowable values (for example, free memory dropping below a critical threshold).
- Online insertion or removal (OIR) of a modular services card (MSC).
- Termination of a process.

The EEM relies on software agents or event detectors to notify it when certain system events occur. When the EEM has detected an event, it can initiate corrective actions. Actions are prescribed in routines called *policies*. Policies must be registered before an action can be applied to collected events. No action occurs unless a policy is registered. A registered policy informs the EEM about a particular event that is to be detected and the corrective action to be taken if that event is detected. When such an event is detected, the EEM enables the corresponding policy. You can disable a registered policy at any time.

The EEM monitors the reliability rates achieved by each process in the system, allowing the system to detect the components that compromise the overall reliability or availability.

This module describes the new and revised tasks you need to configure and manage EEM policies on your the Cisco IOS XR Software network and write and customize the EEM policies using Tool Command Language (Tcl) scripts to handle Cisco IOS XR Software faults and events.

Note For complete descriptions of the event management commands listed in this module, see the Related Documents, on page 53 section of this module.

Feature History for Configuring and Managing Embedded Event Manager Policies

Release	Modification
Release 2.0	This feature was introduced.

Release	Modification
Release 3.3.0	Support was added for AAA authentication when executing a script.
Release 3.6.0	Fault management was replaced with the Embedded Event Manager (EEM) feature. A section on writing and customizing EEM policies using Tcl scripts was added. The None event detector was supported.
Release 3.8.0	Support for the **action_setver_prior** command extension was removed.

Prerequisites for Configuring and Managing Embedded Event Manager Policies

You must be in a user group associated with a task group that includes the proper task IDs. The command reference guides include the task IDs required for each command. If you suspect user group assignment is preventing you from using a command, contact your AAA administrator for assistance.

Information About Configuring and Managing Embedded Event Manager Policies

Event Management

Embedded Event Management (EEM) in the Cisco IOS XR Software system essentially involves system event management. An event can be any significant occurrence (not limited to errors) that has happened within the system. The Cisco IOS XR Software EEM detects those events and implements appropriate responses. The EEM can also be used to prevent or contain faults and to assist in fault recovery.

The EEM enables a system administrator to specify appropriate action based on the current state of the system. For example, a system administrator can use EEM to request notification by e-mail when a hardware device needs replacement.

The EEM also maintains reliability metrics for each process in the system.

System Event Detection

The EEM interacts with routines, “event detectors,” that actively monitor the system for events. The EEM relies on an event detector that it has provided to syslog to detect that a certain system event has occurred. It

uses a pattern match with the syslog messages. It also relies on a timer event detector to detect that a certain time and date has occurred.

Policy-Based Event Response

When the EEM has detected an event, it can initiate actions in response. These actions are contained in routines called *policy handlers*. While the data for event detection is collected, no action occurs unless a policy for responding to that event has been *registered*. At registration, a policy informs the EEM that it is looking for a particular event. When the EEM detects the event, it enables the policy.

Reliability Metrics

The EEM monitors the reliability rates achieved by each process in the system. These metrics can be used during testing to determine which components do not meet their reliability or availability goals so that corrective action can be taken.

System Event Processing

When the EEM receives an event notification, it takes these actions:

- Checks for established policy handlers:
  - If a policy handler exists, the EEM initiates callback routines (*EEM handlers*) or runs Tool Command Language (Tcl) scripts (*EEM scripts*) that implement policies. The policies can include built-in EEM actions.
  - If a policy handler does not exist, the EEM does nothing.
- Notifies the processes that have *subscribed* for event notification.

Note A difference exists between scripts with policy actions and scripts that subscribe to receive events. Scripts with policy actions are expected to implement a policy. They are bound by a rule to prevent recursion. Scripts that subscribe to notifications are not bound by such a rule.

- Records reliability metric data for each process in the system.
- Provides access to EEM-maintained system information through an application program interface (API).

Embedded Event Manager Management Policies

When the EEM has detected an event, it can initiate corrective actions. Actions are prescribed in routines called *policies*. Policies are defined by Tcl scripts (EEM scripts) written by the user through a Tcl API. (See the Embedded Event Manager Scripts and the Scripting Interface (Tcl), on page 4.) Policies must be registered before any action can be applied to collected events. No action occurs unless a policy is registered. A registered policy informs the EEM about a particular event to detect and the corrective action to take if that event is detected. When such an event is detected, the EEM runs the policy. You can disable a registered policy at any time.

Embedded Event Manager Scripts and the Scripting Interface (Tcl)

EEM scripts are used to implement policies when an EEM event is published. EEM scripts and policies are identified to the EEM using the **event manager policy** configuration command. An EEM script remains available to be scheduled by the EEM until the **no event manager policy** command is entered.

The EEM uses these two types of EEM scripts:

- *Regular* EEM scripts identified to the EEM through the **eem script** CLI command. Regular EEM scripts are standalone scripts that incorporate the definition of the event they will handle.
- *EEM callback* scripts identified to the EEM when a process or EEM script registers to handle an event. EEM callback scripts are essentially named functions that are identified to the EEM through the C Language API.

This example shows the usage for the CLI in scripts:

```
sjc-cde-010:/tftpboot/cnwei/fm> cat test_cli_eem.tcl
::cisco::eem::event_register_syslog occurs 1 pattern $_syslog_pattern maxrun 90

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

set errorInfo ""

# 1. query the information of latest triggered fm 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) "config"} 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
}

action_syslog priority info msg "Ran config command $_config_cmd1 $_config_cmd2
```

Script Language

The scripting language is Tool Command Language (Tcl) as implemented within the Cisco IOS XR Software. All Embedded Event Manager scripts are written in Tcl. This full Tcl implementation has been extended by Cisco, and an **eem** command has been added to provide the interface between Tcl scripts and the EEM.

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.

Regular Embedded Event Manager Scripts

Regular EEM scripts are used to implement policies when an EEM event is published. EEM scripts are identified to the EEM using the **event manager policy** configuration command. An EEM script remains available to be scheduled by the EEM until the **no event manager policy** command is entered.

The first executable line of code within an EEM script must be the **eem event register** keyword. This keyword identifies the EEM event for which that script should be scheduled. The keyword is used by the **event manager policy** configuration command to register to handle the specified EEM event.

EEM scripts may use any of the EEM script services listed in Embedded Event Manager Policy Tcl Command Extension Categories, on page 6.

When an EEM script exits, it is responsible for setting a return code that is used to tell the EEM whether to run the default action for this EEM event (if any) or no other action. If multiple event handlers are scheduled for a given event, the return code from the previous handler is passed into the next handler, which can leave the value as is or update it.

Note An EEM script cannot register to handle an event other than the event that caused it to be scheduled.

Embedded Event Manager Callback Scripts

EEM callback scripts are entered as a result of an EEM event being raised for a previously registered EEM event that specifies the name of this script in the eem_handler_spec.

When an EEM callback script exits, it is responsible for setting a return code that is used to tell the EEM whether or not to run the default action for this EEM event (if any). If multiple event handlers are scheduled for a given event, the return code from the previous handler is passed into the next handler, which can leave the value as is or update it.

Note EEM callback scripts are free to use any of the EEM script services listed in Table 1: Embedded Event Manager Tcl Command Extension Categories, on page 6, except for the **eem event register** keyword, which is not allowed in an EEM callback script.

Embedded Event Manager Policy Tcl Command Extension Categories

This table lists the 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: Embedded Event Manager Tcl Command Extension Categories

Category	Definition
EEM event Tcl command extensions(three types: event information, event registration, and event publish)	These Tcl command extensions are 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	These Tcl command extensions are represented by the **sys_reqinfo**_xxx family of system-specific information commands. These commands are used by a policy to gather system information.

Category	Definition
EEM context Tcl command extensions	These Tcl command extensions are used to store and retrieve a Tcl context (the visible variables and their values).

Cisco File Naming Convention for Embedded Event Manager

All EEM 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, EEM policy filenames adhere to the following specifications:

- 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 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.

EEM 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.

EEM 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

Two-Character Abbreviation	Specification
ap	event_register_appl
ct	event_register_counter
st	event_register_stat
no	event_register_none
oi	event_register_oir
pr	event_register_process
sl	event_register_syslog
tm	event_register_timer
ts	event_register_timer_subscriber
wd	event_register_wdsysmon

Embedded Event Manager Built-in Actions

EEM built-in actions can be requested from EEM handlers when the handlers run.

This table describes each EEM handler request or action.

Table 3: Embedded Event Manager Built-In Actions

Embedded Event Manager Built-In Action	Description
Log a message to syslog	Sends a message to the syslog. Arguments to this action are priority and the message to be logged.
Execute a CLI command	Writes the command to the specified channel handler to execute the command by using the **cli_exec** command extension.
Generate a syslog message	Logs a message by using the **action_syslog** Tcl command extension.
Manually run an EEM policy	Runs an EEM policy within a policy while the **event manager run** command is running a policy in EXEC mode.
Publish an application-specific event	Publishes an application-specific event by using the **event_publish appl** Tcl command extension.
Reload the Cisco IOS software	Causes a router to be reloaded by using the EEM **action_reload** command.
Request system information	Represents the sys_reqinfo_xxx family of system-specific information commands by a policy to gather system information.
Send a short e-mail	Sends the e-mail out using Simple Mail Transfer Protocol (SMTP).
Set or modify a counter	Modifies a counter value.

EEM handlers require the ability to run CLI commands. A command is available to the Tcl shell to allow execution of CLI commands from within Tcl scripts.

Application-specific Embedded Event Management

Any Cisco IOS XR Software application can define and publish application-defined events. Application-defined events are identified by a name that includes both the component name and event name, to allow application developers to assign their own event identifiers. Application-defined events can be raised by a Cisco IOS XR Software component even when there are no subscribers. In this case, the EEM dismisses the event, which allows subscribers to receive application-defined events as needed.

An EEM script that subscribes to receive system events is processed in the following order:

1. This CLI configuration command is entered: **event manager policy** *scriptfilename* **username** *username*.
2. The EEM scans the EEM script looking for an **eem event event_type** keyword and subscribes the EEM script to be scheduled for the specified event.
3. The Event Detector detects an event and contacts the EEM.
4. The EEM schedules event processing, causing the EEM script to be run.
5. The EEM script routine returns.

Event Detection and Recovery

Events are detected by routines called *event detectors*. Event detectors are separate programs that provide an interface between other Cisco IOS XR Software components and the EEM. They process information that can be used to publish events, if necessary.

These event detectors are supported:

An EEM event is defined as a notification that something significant has happened within the system. Two categories of events exist:

- System EEM events
- Application-defined events

System EEM events are built into the EEM and are grouped based on the fault detector that raises them. They are identified by a symbolic identifier defined within the API.

Some EEM system events are monitored by the EEM whether or not an application has requested monitoring. These are called *built-in* EEM events. Other EEM events are monitored only if an application has requested EEM event monitoring. EEM event monitoring is requested through an EEM application API or the EEM scripting interface.

Some event detectors can be distributed to other hardware cards within the same secure domain router (SDR) or within the administration plane to provide support for distributed components running on those cards.

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 relationship is between the EEM server, the core event publishers (event detectors), and the event subscribers (policies). 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 —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.

System Manager Event Detector

The System Manager Event Detector has four roles:

- Records process reliability metric data.
- Screens for processes that have EEM event monitoring requests outstanding.
- Publishes events for those processes that match the screening criteria.
- Asks the System Manager to perform its default action for those events that do not match the screening criteria.

The System Manager Event Detector interfaces with the System Manager to receive process startup and termination notifications. The interfacing is made through a private API available to the System Manager. To

minimize overhead, a portion of the API resides within the System Manager process space. When a process terminates, the System Manager invokes a helper process (if specified in the process.startup file) before calling the Event Detector API.

Processes can be identified by component ID, System Manager assigned job ID, or load module pathname plus process instance ID. POSIX wildcard filename pattern support using *, ?, or [...] is provided for load module pathnames. Process instance ID is an integer assigned to a process to differentiate it from other processes with the same pathname. The first instance of a process is assigned an instance ID value of 1, the second 2, and so on.

The System Manager Event Detector handles EEM event monitoring requests for the EEM events shown in this table.

Table 4: System Manager Event Detector Event Monitoring Requests

Embedded Event Manager Event	Description
Normal process termination EEM event—built in	Occurs when a process matching the screening criteria terminates.
Abnormal process termination EEM event—built in	Occurs when a process matching the screening criteria terminates abnormally.
Process startup EEM event—built in	Occurs when a process matching the screening criteria starts.

When System Manager Event Detector abnormal process termination events occur, the default action restarts the process according to the built-in rules of the System Manager.

The relationship between the EEM and System Manager is strictly through the private API provided by the EEM to the System Manager for the purpose of receiving process start and termination notifications. When the System Manager calls the API, reliability metric data is collected and screening is performed for an EEM event match. If a match occurs, a message is sent to the System Manager Event Detector. In the case of abnormal process terminations, a return is made indicating that the EEM handles process restart. If a match does not occur, a return is made indicating that the System Manager should apply the default action.

Timer Services Event Detector

The Timer Services Event Detector implements time-related EEM events. These events are identified through user-defined identifiers so that multiple processes can await notification for the same EEM event.

The Timer Services Event Detector handles EEM event monitoring requests for the Date/Time Passed EEM event. This event occurs when the current date or time passes the specified date or time requested by an application.

Syslog Event Detector

The syslog Event Detector implements syslog message screening for syslog EEM events. This routine interfaces with the syslog daemon through a private API. To minimize overhead, a portion of the API resides within the syslog daemon process.

Screening is provided for the message severity code or the message text fields. POSIX regular expression pattern support is provided for the message text field.

The Syslog Event Detector handles EEM event monitoring requests for the events are shown in this table.

Table 5: Syslog Event Detector Event Monitoring Requests

Embedded Event Manager Event	Description
Syslog message EEM event	Occurs for a just-logged message. It occurs when there is a match for either the syslog message severity code or the syslog message text pattern. Both can be specified when an application requests a syslog message EEM event.
Process event manager EEM event—built in	Occurs when the event-processed count for a specified process is either greater than or equal to a specified maximum or is less than or equal to a specified minimum.

None Event Detector

The None Event Detector publishes an event when the Cisco IOS XR Software **event manager run** CLI command executes an EEM policy. EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. An EEM policy must be identified and registered to be permitted to run manually before the **event manager run** command will execute.

Event manager none detector provides user the ability to run a tcl script using the CLI. The script is registered first before running. Cisco IOS XR Software version provides similar syntax with Cisco IOS EEM (refer to the applicable EEM Documentation for details), so scripts written using Cisco IOS EEM is run on Cisco IOS XR Software with minimum change.

Watchdog System Monitor Event Detector

Watchdog System Monitor (IOSXRWDSysMon) Event Detector for Cisco IOS XR Software

The Cisco IOS XR Software Watchdog System Monitor Event Detector publishes an event when one of the following occurs:

- CPU utilization for a Cisco IOS XR Software process crosses a threshold.
- Memory utilization for a Cisco IOS XR Software process crosses a threshold.

Note Cisco IOS XR Software processes are used to distinguish them from Cisco IOS XR Software Modularity processes.

Two events may be monitored at the same time, and the event publishing criteria can be specified to require one event or both events to cross their specified thresholds.

The Cisco IOS XR Software Watchdog System Monitor Event Detector handles the events as shown in this table.

Table 6: Watchdog System Monitor Event Detector Requests

Embedded Event Manager Event	Description
Process percent CPU EEM event—built in	Occurs when the CPU time for a specified process is either greater than or equal to a specified maximum percentage of available CPU time or is less than or equal to a specified minimum percentage of available CPU time.
Total percent CPU EEM event—built in	Occurs when the CPU time for a specified processor complex is either greater than or equal to a specified maximum percentage of available CPU time or is less than or equal to a specified minimum percentage of available CPU time.
Process percent memory EEM event—built in	Occurs when the memory used for a specified process has either increased or decreased by a specified value.
Total percent available Memory EEM event—built in	Occurs when the available memory for a specified processor complex has either increased or decreased by a specified value.
Total percent used memory EEM event—built in	Occurs when the used memory for a specified processor complex has either increased or decreased by a specified value.

Watchdog System Monitor (WDSysMon) Event Detector for Cisco IOS XR Software Modularity

The Cisco IOS XR Software Software Modularity Watchdog System Monitor Event Detector detects infinite loops, deadlocks, and memory leaks in Cisco IOS XR Software Modularity processes.

Distributed Event Detectors

Cisco IOS XR Software components that interface to EEM event detectors and that have substantially independent implementations running on a distributed hardware card should have a distributed EEM event detector. The distributed event detector permits scheduling of EEM events for local processes without requiring that the local hardware card to the EEM communication channel be active.

These event detectors run on a Cisco IOS XR Software line card:

- System Manager Fault Detector
- Wdsysmon Fault Detector
- Counter Event Detector
- OIR Event Detector
- Statistic Event Detector

Embedded Event Manager Event Scheduling and Notification

When an EEM handler is scheduled, it runs under the context of the process that creates the event request (or for EEM scripts under the Tcl shell process context). For events that occur for a process running an EEM

handler, event scheduling is blocked until the handler exits. The defined default action (if any) is performed instead.

The EEM Server maintains queues containing event scheduling and notification items across client process restarts, if requested.

Reliability Statistics

Reliability metric data for the entire processor complex is maintained by the EEM. The data is periodically written to checkpoint.

Hardware Card Reliability Metric Data

Reliability metric data is kept for each hardware card in a processor complex. Data is recorded in a table indexed by disk ID.

Data maintained by the hardware card is as follows:

- Most recent start time
- Most recent normal end time (controlled switchover)
- Most recent abnormal end time (asynchronous switchover)
- Most recent abnormal type
- Cumulative available time
- Cumulative unavailable time
- Number of times hardware card started
- Number of times hardware card shut down normally
- Number of times hardware card shut down abnormally

Process Reliability Metric Data

Reliability metric data is kept for each process handled by the System Manager. This data includes standby processes running on either the primary or backup hardware card. Data is recorded in a table indexed by hardware card disk ID plus process pathname plus process instance for those processes that have multiple instances.

Process terminations include the following cases:

- Normal termination—Process exits with an exit value equal to 0.
- Abnormal termination by process—Process exits with an exit value not equal to 0.
- Abnormal termination by QNX—Neutrino operating system terminates the process.
- Abnormal termination by kill process API—API kill process terminates the process.

Data to be maintained by process is as follows:

- Most recent process start time
- Most recent normal process end time

- Most recent abnormal process end time
- Most recent abnormal process end type
- Previous ten process end times and types
- Cumulative process available time
- Cumulative process unavailable time
- Cumulative process run time (the time when the process is actually running on the CPU)
- Number of times started
- Number of times ended normally
- Number of times ended abnormally
- Number of abnormal failures within the past 60 minutes
- Number of abnormal failures within the past 24 hours
- Number of abnormal failures within the past 30 days

How to Configure and Manage Embedded Event Manager Policies

Configuring Environmental Variables

EEM environmental variables are Tcl global variables that are defined external to the policy before the policy is run. The EEM policy engine receives notifications when faults and other events occur. EEM policies implement recovery, based on the current state of the system and actions specified in the policy for a given event. Recovery actions are triggered when the policy is run.

Environment Variables

By convention, the names of all environment variables defined by Cisco begin with an underscore character to set them apart; for example, _show_cmd.

Spaces may be used in the *var-value* argument of the **event manager environment** command. The command interprets everything after the *var-name* argument to the end of the line to be part of the *var-value* argument.

Use the **show event manager environment** command to display the name and value of all EEM environment variables after they have been set using the **event manager environment** command.

SUMMARY STEPS

1. **show event manager environment**
2. **configure**
3. **event manager environment** *var-name var-value*
4. Repeat Step 3 for every environment value to be reset.
5. Use the **commit** or **end** command.

6. **show event manager environment**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager environment** **Example:** `RP/0/RP0/CPU0:router# show event manager environment`	Displays the names and values of all EEM environment variables.
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**event manager environment** *var-name var-value* **Example:** `RP/0/RP0/CPU0:router(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7`	Resets environment variables to new values. • The *var-name* argument is the name assigned to the EEM environment configuration variable. • The *var-value* argument is the series of characters, including embedded spaces, to be placed in the environment variable *var-name*. • By convention, the names of all environment variables defined by Cisco begin with an underscore character to set them apart; for example, _show_cmd. • Spaces may be used in the *var-value* argument. The command interprets everything after the *var-name* argument to the end of the line to be part of the *var-value* argument.
Step 4	Repeat Step 3 for every environment value to be reset.	—
Step 5	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

	Command or Action	Purpose
Step 6	**show event manager environment** **Example:** `RP/0/RP0/CPU0:router# show event manager environment`	Displays the reset names and values of all EEM environment variables; allows you to verify the environment variable names and values set in Step 3.

What to do next

After setting up EEM environment variables, find out what policies are available to be registered and then register those policies, as described in the Registering Embedded Event Manager Policies, on page 16.

Registering Embedded Event Manager Policies

Register an EEM policy to run a policy when an event is triggered.

Embedded Event Manager Policies

Registering an EEM policy is performed with the **event manager policy** command in global configuration mode. An EEM script is available to be scheduled by the EEM until the **no** form of this command is entered. Prior to registering a policy, display EEM policies that are available to be registered with the **show event manager policy available** command.

The EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. When the **event manager policy** command is invoked, the EEM examines the policy and registers it to be run when the specified event occurs.

Username

To register an EEM policy, you must specify the username that is used to run the script. This name can be different from the user who is currently logged in, but the registering user must have permissions that are a superset of the username that will run the script. Otherwise, the script is not registered and the command is rejected. In addition, the username that will run the script must have access privileges to the commands run by the EEM policy being registered.

Note AAA authorization (such as the **aaa authorization eventmanager** command) must be configured before EEM policies can be registered. See the *Configuring AAA Services* module of *Configuring AAA Services on Cisco IOS XR Software* for more information about AAA authorization configuration.

Persist-time

An optional **persist-time** keyword for the username can also be defined. The **persist-time** keyword defines the number of seconds the username authentication is valid. When a script is first registered, the configured username for the script is authenticated. After the script is registered, the username is authenticated again each time a script is run. If the AAA server is down, the username authentication can be read from memory. The **persist-time** keyword determines the number of seconds this username authentication is held in memory.

- If the AAA server is down and the **persist-time** keyword has not expired, then the username is authenticated from memory and the script runs.

- If the AAA server is down, and the **persist-time** keyword has expired, then user authentication will fail and the script will not run.

The following values can be used for the **persist-time** keyword.

- The default **persist-time** is 3600 seconds (1 hour). Enter the **event manager policy** command without the **persist-time** keyword to set the **persist-time** to 1 hour.
- Enter 0 to stop the username authentication from being cached. If the AAA server is down, the username will not authenticate and the script will not run.
- Enter **infinite** to stop the username from being marked as invalid. The username authentication held in the cache will not expire. If the AAA server is down, the username will be authenticated from the cache.

System or user keywords

If you enter the **event manager policy** command without specifying either the **system** or **user** keyword, the EEM first tries to locate the specified policy file in the system policy directory. If the EEM finds the file in the system policy directory, it registers the policy as a system policy. If the EEM does not find the specified policy file in the system policy directory, it looks in the user policy directory. If the EEM locates the specified file in the user policy directory, it registers the policy file as a user policy. If the EEM finds policy files with the same name in both the system policy directory and the user policy directory, the policy file in the system policy directory takes precedence and is registered as a system policy.

Once policies have been registered, their registration can be verified through the **show event manager policy registered** command. The output displays registered policy information in two parts. The first line in each policy description lists the index number assigned to the policy, the policy type (system or user), the type of event registered, the time when the policy was registered, and the name of the policy file. The remaining lines of each policy description display information about the registered event and how the event is to be handled, and come directly from the Tcl command arguments that make up the policy file.

SUMMARY STEPS

1. **show event manager policy available** [**system** | **user**]
2. **configure**
3. **event manager policy** *policy-name* **username** *username* [**persist-time** { *seconds* | **infinite** }] | **type** { **system** | **user** }
4. Repeat Step 3 for every EEM policy to be registered.
5. Use the **commit** or **end** command.
6. **show event manager policy registered**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy available** [**system**	**user**] **Example:** `RP/0/RP0/CPU0:router# show event manager policy available`	Displays all EEM policies that are available to be registered. • Entering the optional **system** keyword displays all available system policies. • Entering the optional **user** keyword displays all available user policies.

	Command or Action	Purpose
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.

Step 3	**event manager policy** *policy-name* **username** *username* [**persist-time** { *seconds* | **infinite** }] | **type** { **system** | **user** }
	Example:
	`RP/0/RP0/CPU0:router(config)# event manager policy cron.tcl username tom type user`
		Registers an EEM policy with the EEM.
		• An EEM script is available to be scheduled by the EEM until the **no** form of this command is entered.
		• Enter the required **username** keyword and argument, where *username* is the username that runs the script.
		• Enter the optional **persist-time** keyword to determine how long the username authentication is held in memory:
		  • Enter the number of *seconds* for the **persist-time** keyword.
		  • Enter the **infinite** keyword to make the authentication permanent (the authentication will not expire).
		• Entering the optional **type system** keywords registers a system policy defined by Cisco.
		• Entering the optional **type user** keywords registers a user-defined policy.
		Note AAA authorization (such as aaa authorization eventmanager) must be configured before EEM policies can be registered. See the *Configuring AAA Services* module of *System Security Configuration Guide for Cisco CRS Routers* for more information about AAA authorization configuration.

Step 4	Repeat Step 3 for every EEM policy to be registered.	—
Step 5	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

	Command or Action	Purpose
Step 6	**show event manager policy registered** **Example:** `RP/0/RP0/CPU0:router# show event manager policy registered`	Displays all EEM policies that are already registered, allowing verification of Step 3.

How to Write Embedded Event Manager Policies Using Tcl

This section provides information on how to write and customize Embedded Event Manager (EEM) policies using Tool Command Language (Tcl) scripts to handle Cisco IOS XR Software faults and events.

This section contains these tasks:

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

A policy must be available that is written in the Tcl scripting language. Sample policies are provided in the Sample EEM Policies, on page 25. Sample policies are stored in the system policy directory.

SUMMARY STEPS

1. **show event manager environment** [**all** | *environment-name*]
2. **configure**
3. **event manager environment** *var-name* [*var-value*]
4. Repeat Step 3, on page 20 to configure all the environment variables required by the policy to be registered in Step 5, on page 20.
5. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
6. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager environment** [**all**	*environment-name*] **Example:** `RP/0/RP0/CPU0:router# show event manager environment all`	(Optional) Displays the name and value of EEM environment variables. • The **all** keyword displays all the EEM environment variables. • The *environment-name* argument displays information about the specified environment variable.

	Command or Action	Purpose
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**event manager environment** *var-name* [*var-value*] **Example:** `RP/0/RP0/CPU0:router(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7`	Resets environment variables to new values. • The *var-name* argument is the name assigned to the EEM environment configuration variable. • The *var-value* argument is the series of characters, including embedded spaces, to be placed in the environment variable *var-name* . • By convention, the names of all environment variables defined by Cisco begin with an underscore character to set them apart; for example, _show_cmd. • Spaces may be used in the *var-value* argument. The command interprets everything after the *var-name* argument to the end of the line to be part of the *var-value* argument.
Step 4	Repeat Step 3, on page 20 to configure all the environment variables required by the policy to be registered in Step 5, on page 20.	—
Step 5	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] **Example:** `RP/0/RP0/CPU0:router(config)# event manager policy tm_cli_cmd.tcl username user_a 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 system policy defined by Cisco. • Use the **user** keyword to register a user-defined system policy. • Use the **persist-time** keyword to specify the length of time the username authentication is valid. In this example, the sample EEM policy named tm_cli_cmd.tcl is registered as a system policy.
Step 6	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes.

	Command or Action	Purpose
		• **Cancel** —Remains in the configuration session, without committing the configuration changes.

Displaying EEM Registered Policies

Perform this optional task to display EEM registered policies.

SUMMARY STEPS

1. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy registered** [**event-type** *type*] [**system**	**user**] [**time-ordered**	**name-ordered**] **Example:** `RP/0/RP0/CPU0:router# show event manager policy registered system`	Displays information about currently registered policies. • The **event-type** keyword displays the registered policies for a specific event type. • The **time-ordered** keyword displays information about currently registered policies sorted by time. • The **name-ordered** keyword displays the policies in alphabetical order by the policy name.

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. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]
2. **configure**
3. **no event manager policy** *policy-name*
4. Use the **commit** or **end** command.
5. Repeat Step 1, on page 21to ensure that the policy has been removed.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy registered** [**event-type** *type*] [**system**	**user**] [**time-ordered**	**name-ordered**] **Example:**	Displays information about currently registered policies. • The **event-type** keyword displays the registered policies for a specific event type.

	Command or Action	Purpose
	`RP/0/RP0/CPU0:router# show event manager policy registered system`	• The **time-ordered** keyword displays information about currently registered policies sorted by time. • The **name-ordered** keyword displays the policies in alphabetical order by the policy name.
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**no event manager policy** *policy-name* **Example:** `RP/0/RP0/CPU0:router(config)# no event manager policy tm_cli_cmd.tcl`	Removes the EEM policy from the configuration, causing the policy to be unregistered.
Step 4	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.
Step 5	Repeat Step 1, on page 21to ensure that the policy has been removed.	—

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. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]
2. **configure**
3. **event manager scheduler suspend**
4. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**] **Example:** `RP/0/RP0/CPU0:router# show event manager policy registered system`	Displays information about currently registered policies. • The **event-type** keyword displays the registered policies for a specific event type. • The **time-ordered** keyword displays information about currently registered policies sorted by time. • The **name-ordered** keyword displays the policies in alphabetical order by the policy name.
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**event manager scheduler suspend** **Example:** `RP/0/RP0/CPU0:router(config)# event manager scheduler suspend`	Immediately suspends the execution of all EEM policies.
Step 4	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

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. **show event manager directory user** [**library** | **policy**]
2. **configure**
3. **event manager directory user** {**library** *path* | **policy** *path*}

4. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager directory user** [**library** | **policy**] **Example:** `RP/0/RP0/CPU0:router# show event manager directory user library`	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 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**event manager directory user** {**library** *path* | **policy** *path*} **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user library disk0:/usr/lib/tcl`	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 4	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

Displaying Software Modularity Process Reliability Metrics Using EEM

Perform this optional task to display reliability metrics for Cisco IOS XR Software processes.

SUMMARY STEPS

1. **show event manager metric process** {**all** | *job-id* | *process-name*} **location** {**all** | *node-id*}

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager metric process** {**all** | *job-id* | *process-name*} **location** {**all** | *node-id*} **Example:** `RP/0/RP0/CPU0:router# show event manager environment`	Displays 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.

Sample EEM Policies

Cisco IOS XR Software contains some sample policies in the images that contain the EEM. 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.

The Cisco IOS XR Software includes a set of sample policies (see *Sample EEM Policy Descriptions* table). The sample policies can be copied to a user directory and then modified. Tcl is currently the only scripting language supported by Cisco 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 20 seconds.

Sample EEM policies can be seen on the router using the CLI

```
Show event manager policy available system
```

This table describes the sample EEM policies.

Table 7: Sample EEM Policy Descriptions

Name of Policy	Description
periodic_diag_cmds.tcl	This policy is triggered when the _cron_entry_diag cron entry expires.Then, the output of this fixed set is collect for the fixed set of commands and the output is sent by email.
periodic_proc_avail.tcl	This policy is triggered when the _cron_entry_procavail cron entry expires. Then the output of this fixed set is collect for the fixed set of commands and the output is sent by email.
periodic_sh_log.tcl	This policy is triggered when the _cron_entry_log cron entry expires, and collects the output for the show log command and a few other commands. If the environment variable _log_past_hours is configured, it collects the log messages that are generated in the last _log_past_hours hours. Otherwise, it collects the full log.
sl_sysdb_timeout.tcl	This policy is triggered when the script looks for the sysdb timeout ios_msgs and obtains the output of the show commands. The output is written to a file named after the blocking process.
tm_cli_cmd.tcl	This policy runs using a configurable CRON entry. It executes a configurable CLI command and e-mails the results.

Name of Policy	Description
tm_crash_hist.tcl	This policy runs at midnight each day and e-mails a process crash history report to a specified e-mail address.

For more details about the sample policies available and how to run them, see the EEM Event Detector Demo: Example , on page 43.

SUMMARY STEPS

1. **show event manager policy available** [**system** | **user**]
2. **configure**
3. **event manager directory user** {**library** *path* | **policy** *path*}
4. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
5. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy available** [**system**	**user**] **Example:** `RP/0/RP0/CPU0:router# show event manager policy available`	Displays EEM policies that are available to be registered.
Step 2	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 3	**event manager directory user** {**library** *path*	**policy** *path*} **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user library disk0:/user_library`	Specifies a directory to use for storing user library files or user-defined EEM policies.
Step 4	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds*	**infinite**]	**type** [**system**	**user**]] **Example:** `RP/0/RP0/CPU0:router(config)# event manager policy test.tcl username user_a type user`	Registers the EEM policy to be run when the specified event defined within the policy occurs.
Step 5	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session.

	Command or Action	Purpose
		• **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

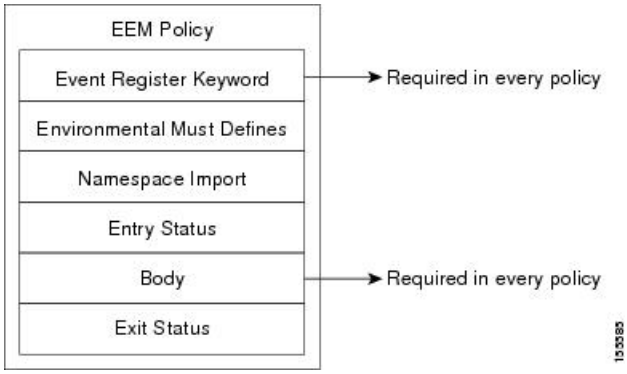
Programming EEM Policies with Tcl

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 27 are optional.

Tcl Policy Structure and Requirements

All EEM policies share the same structure, shown in Figure 1: Tcl Policy Structure and Requirements , on page 27. 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: environmental must defines, namespace import, entry status, and exit status.

Figure 1: 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. For a list of the available EEM **event_register** Tcl command extensions, see the Embedded Event Manager Event Registration Tcl Command Extensions, on page 54. 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 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. For a list of the available EEM event information Tcl command extensions, see the Embedded Event Manager Event Information Tcl Command Extension, on page 78.
- The action Tcl command extensions, such as **action_syslog**, that are used to specify actions specific to EEM. For a list of the available EEM action Tcl command extensions, see the Embedded Event Manager Action Tcl Command Extensions, on page 97.
- The system information Tcl command extensions, such as **sys_reqinfo_routername**, that are used to obtain general system information. For a list of the available EEM system information Tcl command extensions, see the Embedded Event Manager System Information Tcl Command Extensions, on page 113.
- Use of the SMTP library (to send e-mail notifications) or the CLI library (to run CLI commands) from a policy. For a list of the available SMTP library Tcl command extensions, see the SMTP Library Command Extensions, on page 124. For a list of the available CLI library Tcl command extensions, see the CLI Library Command Extensions, on page 126.
- The **context_save** and **con text_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 to 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),
- 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 EEM to determine whether or not to apply the default action for this event, if any. A value of zero means that the default action should not be performed. A value of nonzero means that the default action should be performed. The exit status is 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, the 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 following example 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 this table.

Table 8: _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 preceding case, which indicates CERR_CLASS_WARNING: See Table 9: Error Class Encodings, on page 30 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.
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 this table.

Table 9: Error Class Encodings

Error Return Value	Error Class
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. **show event manager policy available** [**system** | **user**]
2. Cut and paste the contents of the sample policy displayed on the screen to a text editor.
3. Define the required event_register Tcl command extension.
4. Add the appropriate namespace under the ::cisco hierarchy.
5. Program the must defines section to check for each environment variable that is used in this policy.
6. Program the body of the script.
7. Check the entry status to determine if a policy has previously run for this event.

8. Check the exit status to determine whether or not to apply the default action for this event, if a default action exists.
9. Set Cisco Error Number (_cerrno) Tcl global variables.
10. Save the Tcl script with a new filename, and copy the Tcl script to the router.
11. **configure**
12. **event manager directory user** {**library** *path* | **policy** *path*}
13. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
14. Use the **commit** or **end** command.
15. Cause the policy to execute, and observe the policy.
16. Use debugging techniques if the policy does not execute correctly.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**show event manager policy available** [**system** | **user**] **Example:** `RP/0/RP0/CPU0:router# show event manager policy available`	Displays EEM policies that are available to be registered.
Step 2	Cut and paste the contents of the sample policy displayed on the screen to a text editor.	—
Step 3	Define the required event_register Tcl command extension.	Choose the appropriate event_register Tcl command extension for the event that you want to detect, and add it to the policy. The following are valid Event Registration Tcl Command Extensions: • event_register_appl • event_register_counter • event_register_stat • event_register_wdsysmon • event_register_oir • event_register_process • event_register_syslog • event_register_timer • event_register_timer_subscriber • event_register_hardware • event_register_none
Step 4	Add the appropriate namespace under the ::cisco hierarchy.	Policy developers can use the new namespace ::cisco in Tcl policies to group all the extensions used by Cisco IOS XR EEM. There are two namespaces under the ::cisco

	Command or Action	Purpose

		hierarchy. The following are the namespaces and the EEM Tcl command extension categories that belongs under each namespace:
		• ::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 Ensure that the appropriate namespaces are imported, or use the qualified command names when using the preceding commands.

Step 5	Program the must defines section to check for each environment variable that is used in this policy.	This is an optional step. Must defines is 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 EEM configuration command **event manager environment** . By convention, all Cisco EEM environment variables begin with "_" (an underscore). 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** command in EXEC mode. For example, EEM environment variables defined by the sample policies include e-mail variables. The sample policies that send e-mail must have the following variables

	Command or Action	Purpose
		set in order to function properly. The following are the e-mail-specific environment variables used in the sample EEM policies. • **_email_server**—A Simple Mail Transfer Protocol (SMTP) mail server used to send e-mail (for example, mailserver.example.com) • **_email_to**—The address to which e-mail is sent (for example, engineering@example.com) • **_email_from**—The address from which e-mail is sent (for example, devtest@example.com) • **_email_cc**—The address to which the e-mail must be copied (for example, manager@example.com)
Step 6	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 actions specific to EEM. • 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 7	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 8	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 that the default action should not be performed. A value of nonzero means that the default action should be performed. The exit status is passed to subsequent policies that are run for the same event.
Step 9	Set Cisco Error Number (_cerrno) Tcl global variables.	Some EEM Tcl command extensions set a Cisco Error Number Tcl global variable _cerrno. Whenever _cerrno

	Command or Action	Purpose
		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).
Step 10	Save the Tcl script with a new filename, and copy the Tcl script to the router.	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 Table 2: Two-Character Abbreviation Specification, on page 7) 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 theCisco File Naming Convention for Embedded Event Manager, on page 7. Copy the file to the flash file system on the router—typically disk0:.
Step 11	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 12	**event manager directory user** {**library** *path* | **policy** *path*} **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user library disk0:/user_library`	Specifies a directory to use for storing user library files or user-defined EEM policies.
Step 13	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] **Example:** `RP/0/RP0/CPU0:router(config)# event manager policy test.tcl username user_a type user`	Registers the EEM policy to be run when the specified event defined within the policy occurs.
Step 14	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session.

	Command or Action	Purpose
		• **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.
Step 15	Cause the policy to execute, and observe the policy.	—
Step 16	Use debugging techniques if the policy does not execute correctly.	—

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 are not found when an EEM policy that references a Tcl procedure is run.

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 from Step 1, on page 35and the tclIndex file from Step 3, on page 36to the directory used for storing user library files on the target router.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target router.
6. **configure**
7. **event manager directory user library** *path*
8. **event manager directory user policy** *path*
9. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
10. **event manager run** *policy* [*argument*]
11. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose

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
		`proc test1 {} {`
		`  puts "In procedure test1"`
		`}`
		`proc test2 {} {`

	Command or Action	Purpose

		`puts "In procedure test2"`
		`}`
		lib2.tcl
		`proc test3 {} {`
		`puts "In procedure test3"`
		`}`

Step 2	**tclsh** **Example:** `workstation% tclsh`	Enters the Tcl shell.

Step 3	**auto_mkindex** *directory_name* ***.tcl**
	Example:
	`workstation% auto_mkindex eem_library *.tcl`
		Use the **auto_mkindex** command to create the tclIndex file. The tclIndex file 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 be only 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.
		The following sample 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
		`# 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 from Step 1, on page 35and the tclIndex file from Step 3, on page 36to the directory used for storing user library files on the target router.	—
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 router.	The directory can be the same directory used in Step 4, on page 36.

	Command or Action	Purpose

		The following example user-defined EEM policy can be used to test the Tcl library support in EEM:
		libtest.tcl
		`::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	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 7	**event manager directory user library** *path* **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user library disk2:/eem_library`	Specifies the EEM user library directory; this is the directory to which the files in Step 4, on page 36 were copied.
Step 8	**event manager directory user policy** *path* **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user policy disk2:/eem_policies`	Specifies the EEM user policy directory; this is the directory to which the file in Step 5, on page 36was copied.
Step 9	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] **Example:** `RP/0/RP0/CPU0:router(config)# event manager policy libtest.tcl username user_a`	Registers a user-defined EEM policy.
Step 10	**event manager run** *policy* [*argument*] **Example:** `RP/0/RP0/CPU0:router(config)# event manager run libtest.tcl`	Manually runs an EEM policy.

	Command or Action	Purpose
Step 11	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

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 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 the Tcl packages contained in the directory along with version information. If the index is not created, the Tcl packages are not found when an EEM policy that contains a **package require** Tcl command is run.

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 package files from Step 1 and the pkgIndex file from Step 3 to the directory used for storing user library files on the target router.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target router.
6. **configure**
7. **event manager directory user library** *path*
8. **event manager directory user policy** *path*

9. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
10. **event manager run** *policy* [*argument*]
11. Use the **commit** or **end** command.

DETAILED STEPS

	Command or Action	Purpose
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** **Example:** `workstation% tclsh`	Enters the Tcl shell.

Step 3	**pkg_mkindex** *directory_name* ***.tcl**
	Example:
	`workstation% pkg_mkindex eem_library *.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 the **pkg_mkindex** command inside a directory, because there can be only a single pkgIndex file in any directory and you may have other Tcl files to be grouped together. Running the **pkg_mkindex** command in a directory determines which Tcl package file or files are indexed using a specific pkgIndex.
		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
		`# 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 package files from Step 1 and the pkgIndex file from Step 3 to the directory used for storing user library files on the target router.	—

	Command or Action	Purpose

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 router.	The directory can be the same directory used in Step 4, on page 39.
		The following example user-defined EEM policy can be used to test the Tcl library support in EEM:
		packagetest.tcl
		`::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	**configure** **Example:** `RP/0/RP0/CPU0:router# configure`	Enters global configuration mode.
Step 7	**event manager directory user library** *path* **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user library disk2:/eem_library`	Specifies the EEM user library directory; this is the directory to which the files in Step 4, on page 39were copied.
Step 8	**event manager directory user policy** *path* **Example:** `RP/0/RP0/CPU0:router(config)# event manager directory user policy disk2:/eem_policies`	Specifies the EEM user policy directory; this is the directory to which the file in Step 5, on page 40was copied.
Step 9	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] **Example:** `RP/0/RP0/CPU0:router(config)# event manager policy packagetest.tcl username user_a`	Registers a user-defined EEM policy.
Step 10	**event manager run** *policy* [*argument*] **Example:** `RP/0/RP0/CPU0:router(config)# event manager run packagetest.tcl`	Manually runs an EEM policy.

	Command or Action	Purpose
Step 11	Use the **commit** or **end** command.	**commit** —Saves the configuration changes and remains within the configuration session. **end** —Prompts user to take one of these actions: • **Yes** — Saves configuration changes and exits the configuration session. • **No** —Exits the configuration session without committing the configuration changes. • **Cancel** —Remains in the configuration session, without committing the configuration changes.

Configuration Examples for Event Management Policies

Environmental Variables Configuration: Example

This configuration sets the environment variable cron_entry:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router#(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7
```

User-Defined Embedded Event Manager Policy Registration: Example

This configuration registers a user-defined event management policy:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# event manager policy cron.tcl username tom user
```

Display Available Policies: Example

This is the sample output from the **show event manager policy available** command displaying available policies:

```
RP/0/RP0/CPU0:router# show event manager policy available

No.  Type    Time Created                 Name
1    system  Mon Mar 15 21:32:14 2004     periodic_diag_cmds.tcl
2    system  Mon Mar 15 21:32:14 2004     periodic_proc_avail.tcl
3    system  Mon Mar 15 21:32:16 2004     periodic_sh_log.tcl
4    system  Mon Mar 15 21:32:16 2004     tm_cli_cmd.tcl
5    system  Mon Mar 15 21:32:16 2004     tm_crash_hist.tcl
```

Display Embedded Event Manager Process: Example

Reliability metric data is kept for each process handled by the System Manager. This data includes standby processes running on either the primary or backup hardware card. Data is recorded in a table indexed by hardware card disk ID plus process pathname plus process instance for those processes that have multiple instances. This is the sample output from the **show event manager metric process** command displaying reliability metric data:

```
RP/0/RP0/CPU0:router# show event manager metric process all location 0/1/CPU0

====================================
job id: 78, node name: 0/1/CPU0
process name: wd-critical-mon, instance: 1
--------------------------------
last event type: process start
recent start time: Mon Sep 10 21:36:49 2007
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:
--------------------------
Mon Sep 10 21:36:49 2007
--------------------------

most recent 10 process end times and types:

cumulative process available time: 59 hours 33 minutes 42 seconds 638 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  1.000000000
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
====================================
job id: 56, node name: 0/1/CPU0
process name: dllmgr, instance: 1
--------------------------------
last event type: process start
recent start time: Mon Sep 10 21:36:49 2007
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:
--------------------------
Mon Sep 10 21:36:49 2007
--------------------------

most recent 10 process end times and types:

cumulative process available time: 59 hours 33 minutes 42 seconds 633 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  1.000000000
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
====================================
```

Configuration Examples for Writing Embedded Event Manager Policies Using Tcl

EEM Event Detector Demo: Example

This example uses the sample policies to demonstrate how to use Embedded Event Manager policies. Proceed through the following sections to see how to use the sample policies:

EEM Sample Policy Descriptions

The configuration example features one sample EEM policy. The tm_cli_cmd.tcl runs using a configurable CRON entry. This policy executes a configurable CLI command and e-mails the results.

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.

This table describes a list of the e-mail variables.

Table 10: E-mail-Specific Environmental Variables Used by the Sample Policies

Environment Variable	Description	Example
_email_server	Simple Mail Transfer Protocol (SMTP) mail server used to send e-mail.	mailserver.example.com
_email_to	Address to which e-mail is sent.	engineering@example.com
_email_from	Address from which e-mail is sent.	devtest@example.com
_email_cc	Address to which the e-mail must be copied.	manager@example.com

This table 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	First configuration command that is run.	**interface gigabitEthernet1/0/5/0**
_config_cmd2	Second configuration command that is run. This variable is optional and need not be specified.	**no shutdown**

Environment Variable	Description	Example
_syslog_pattern	Regular expression pattern match string that is used to compare syslog messages to determine when the policy runs.	.*UPDOWN.*FastEthernet0/0.*

This table 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	CRON specification that determines when the policy will run.	0-59/1 0-23/1 * * 0-7
_show_cmd	CLI command to be executed when the policy is run.	**show version**

This table 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	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	URL location to which the crash report is sent.	http://www.example.com/fm/interface_tm.cgi

This table 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	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
_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

Environment Variable	Description	Example
_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 (EEM), the **hostname** and **domain-name** commands must be configured. The EEM environment variables must also be set. After a Cisco IOS XR Software 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 Table 14: Environment Variables Used in the tm_fsys_usage.tcl Policy, on page 44) are all optional and are not listed here:

```
hostname cpu
event manager environment _domainname 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 these configuration scenarios to demonstrate how to use the four sample Tcl policies:

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 run 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 EXEC mode, use 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, which displays policies that are available to be installed. After you enter the **configure** 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 run 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 EXEC mode, 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, which displays the policies that are available to be installed. After you enter the **configure** 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 router 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 EXEC mode, enter the **show event manager policy registered** command to verify that no policies are currently registered. Next, enter the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure** 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 **show event manager policy registered** command to verify that no policies are currently registered. Next, enter the **show event manager policy available** command to display which policies are available to be installed. After you enter the **configure** 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 two of the sample policies that are included as EEM system policies. For more details about these policies, see the EEM Event Detector Demo: Example , on page 43.

tm_cli_cmd.tcl Sample Policy

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

```
::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 XR 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                   disk0:/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
}

# 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) "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 preceding EEM sample policy:

```
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
```

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
```

Additional References

The following sections provide references related to configuring and managing Embedded Event Manager policies.

Related Documents

Related Topic	Document Title
Embedded Event Manager commands	*Embedded Event Manager Commands* module in the *System Monitoring Command Reference for Cisco CRS Routers*
Route processor failover commands	Hardware Redundancy and Node Administration Commands module in the *Interface and Hardware Component Command Reference for Cisco CRS Routers*
Cisco IOS XR XML API material	*Cisco IOS XR XML API Guide for the Cisco CRS Router*
Cisco IOS XR getting started material	*Cisco IOS XR Getting Started Guide for the Cisco CRS Router*
Information about user groups and task IDs	*Configuring AAA Services* module in the *System Security Configuration Guide for Cisco CRS Routers*

Standards

Standards	Title
No new or modified standards are supported by this feature, and support for existing standards has not been modified by this feature.	—

MIBs

MIBs	MIBs Link
—	To locate and download MIBs using Cisco IOS XR software, use the Cisco MIB Locator found at the following URL and choose a platform under the Cisco Access Products menu: http://cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml

RFCs

RFCs	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 Technical Support website contains thousands of pages of searchable technical content, including links to products, technologies, solutions, technical tips, and tools. Registered Cisco.com users can log in from this page to access even more content.	*http://www.cisco.com/cisco/web/support/index.html*

Embedded Event Manager Policy Tcl Command Extension Reference

This section documents the following EEM policy Tcl command extension categories:

Note For all EEM Tcl command extensions, if there is an error, the returned Tcl result string contains the error information.

Note Arguments for which no numeric range is specified take an integer from -2147483648 to 2147483647, inclusive.

The following conventions are used for the syntax documented on the Tcl command extension pages:

- An optional argument is shown within square brackets, for example:

```
[type ?]
```

- A question mark ? represents a variable to be entered.
- Choices between arguments are represented by pipes, for example:

```
[queue_priority low|normal|high]
```

Embedded Event Manager Event Registration Tcl Command Extensions

The following EEM event registration Tcl command extensions are supported:

event_register_appl

Registers for an application event. Use this Tcl command extension to run a policy when an application event is triggered following another policy's execution of an event_publish Tcl command extension; the event_publish command extension publishes an application event.

To register for an application event, a subsystem must be specified. Either a Tcl policy or the internal EEM API can publish an application event. If the event is being published by a policy, the *sub_system* argument that is reserved for a policy is 798.

Syntax

```
event_register_appl [sub_system ?] [type ?] [queue_priority low|normal|high] [maxrun ?]
[nice 0|1]
```

Arguments

sub_system	(Optional) Number assigned to the EEM policy that published the application event. The number is set to 798, because all other numbers are reserved for Cisco use. If this argument is not specified, all components are matched.
type	(Optional) Event subtype within the specified event. The *sub_system* and *type* arguments uniquely identify an application event. If this argument is not specified, all types are matched. If you specify this argument, you must choose an integer between 1 and 4294967295, inclusive. There must be a match of component and type between the **event_publish** command extension and the **event_register_appl** command extension for the publishing and registration to work.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

If multiple conditions exist, the application event is raised when all the conditions are satisfied.

Result String

None

Set _cerrno

No

event_register_cli

Registers for a CLI event. Use this Tcl command extension to run a policy when a CLI command of a specific pattern is entered based on pattern matching performed against an expanded CLI command. This will be implemented as a new process in IOS-XR which will be dlrsc_tracker. This ED will not do pattern match on admin commands of XR.

Note You can enter an abbreviated CLI command, such as **sh mem summary**, and the parser will expand the command to **show memory summary** to perform the matching. The functionality provided in the CLI event detector only allows a regular expression pattern match on a valid XR CLI command itself. This does not include text after a pipe character when redirection is used.

Syntax

```
event_register_cli [tag ?]
[occurs ?] [period ?] pattern ? [default ?] [queue_priority low|normal|high|last] [maxrun
?] [nice 0|1]
```

Arguments

tag	(Optional) String identifying a tag that can be used with the trigger Tcl command extension to support multiple event statements within a Tcl script.
occurs	(Optional) The number of occurrences before the event is raised. If this argument is not specified, the event is raised on the first occurrence. If this argument is specified, it must be an integer between 1 and 4294967295, inclusive.
period	(Optional) Specifies a backward looking time window in which all CLI events must occur (the occurs clause must be satisfied) in order for an event to be published (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the most recent event is used.
pattern	(Mandatory) Specifies the regular expression used to perform the CLI command pattern match.
default	(Optional) The time period during which the CLI event detector waits for the policy to exit (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If the default time period expires before the policy exits, the default action will be executed. The default action is to run the command. If this argument is not specified, the default time period is set to 30 seconds.

If multiple conditions are specified, the CLI event will be raised when all the conditions are matched.

Result String

None

Set _cerrno

No

event_register_config

Registers for a change in running configuration. Use this Tcl command extension to trigger a policy when there is any configuration change. This will be implemented as a new process in IOS-XR which will be dlrsc_tracker. This ED will not check for admin config changes in XR.

Syntax

```
event_register_config
[queue_priority low|normal|high|last]
[maxrun ?] [nice 0|1]
```

Arguments

queue_priority	(Optional) Priority level at which the script will be queued: • queue_priority low-Specifies that the script is to be queued at the lowest of the three priority levels. • queue_priority normal-Specifies that the script is to be queued at a priority level greater than low priority but less than high priority. • queue_priority high-Specifies that the script is to be queued at the highest of the three priority levels. • queue_priority last-Specifies that the script is to be queued at the lowest priority level. If more than one script is registered with the "queue_priority_last" argument set, these scripts will execute in the order in which the events are published. **Note** The queue_priority argument specifies the queuing priority, but not the execution priority, of the script being registered. If this argument is not specified, the default queuing priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the nice argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

If multiple conditions are specified, the syslog event will be raised when all the conditions are matched.

Result String

None

Set _cerrno

No

event_register_counter

Registers for a counter event as both a publisher and a subscriber. Use this Tcl command extension to run a policy on the basis of a named counter crossing a threshold. This event counter, as a subscriber, identifies the name of the counter to which it wants to subscribe and depends on another policy or another process to actually manipulate the counter. For example, let policyB act as a counter policy, whereas policyA (although it does not need to be a counter policy) uses register_counter, counter_modify, or unregister_counter Tcl command extensions to manipulate the counter defined in policyB.

Syntax

```
event_register_counter name ? entry_op gt|ge|eq|ne|lt|le entry_val ?
exit_op gt|ge|eq|ne|lt|le exit_val ? [queue_priority low|normal|high]
[maxrun ?] [nice 0|1]
```

Arguments

name	(Mandatory) Name of the counter.
entry_op	(Mandatory) Entry comparison operator used to compare the current counter value with the entry value; if true, an event is raised and event monitoring is disabled until exit criteria are met.
entry_val	(Mandatory) Value with which the current counter value should be compared, to decide if the counter event should be raised.
exit_op	(Mandatory) Exit comparison operator used to compare the current counter value with the exit value; if true, event monitoring for this event is reenabled.
exit_val	(Mandatory) Value with which the current counter value should be compared to decide if the exit criteria are met.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Result String

None

Set _cerrno

No

event_register_hardware

Registers for an environmental monitoring hardware device that is specified by the hardware event and condition.

Syntax

```
event_register_hardware env_device ? env_cond ?
[priority normal|low|high] [maxrun_sec ?] [maxrun_nsec ?] [nice 0|1]
```

Arguments

env_device	(Mandatory) Environmental device that is used to monitor.The integer number must be inclusively between 1 and 2147483647. This is a bit mask that monitors multiple types of environmental devices. The following supported devices and their corresponding bitmasks are listed: • 0x0001 chassis • 0x0002 backplane • 0x0004 slot • 0x0008 card • 0x0010 port • 0x0020 fan • 0x0040 group of power supplies • 0x0080 power supply • 0x0100 sensor They can be bit wise OR'ed to monitor multiple devices.
env_cond	(Mandatory) Environmental condition that is used to monitor. This is a bit mask that monitors multiple kinds of environmental conditions. The following supported environmental conditions and their corresponding bitmasks are listed: • 0x0001 low warning • 0x0002 high warning • 0x0004 warning • 0x0010 low critical • 0x0020 high critical • 0x0040 critical • 0x0100 pre-shutdown • 0x0200 shutdown
priority	(Optional) Priority level that the script is queued. If not specified, the default uses the normal priority.
maxrun_sec, maxrun_nsec	(Optional) Maximum runtime of the script that is specified in seconds and nanoseconds. The integer number must be inclusively between 0 and 2147483647. If not specified, use the default 20-second run-time limit.
nice	(Optional) Maximum runtime of the script that is specified in seconds and nanoseconds. The integer number must be inclusively between 0 and 2147483647. If not specified, use the default 20-second run-time limit.

Result String

None

Set _cerrno

No

event_register_none

Registers for an event that is triggered by the event manager run command. These events are handled by the None event detector that screens for this event.

Syntax

```
event_register_none [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

Arguments

queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Result String

None

Set _cerrno

No

event_register_oir

Registers for an online insertion and removal (OIR) event. Use this Tcl command extension to run a policy on the basis of an event raised when a hardware card OIR occurs. These events are handled by the OIR event detector that screens for this event.

Syntax

```
event_register_oir [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

Arguments

queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Result String

None

Set _cerrno

No

event_register_process

Registers for a process event. Use this Tcl command extension to run a policy on the basis of an event raised when a Cisco IOS XR software modularity process starts or stops. These events are handled by the system manager event detector that screens for this event. This Tcl command extension is supported only in software modularity images.

Syntax

```
event_register_process abort|term|start
[job_id ?] [instance ?] [path ?] [node ?]
[queue_priority low|normal|high] [maxrun ?] [nice 0|1] [tag?]
```

Arguments

abort	(Mandatory) Abnormal process termination. Process may terminate because of exiting with a nonzero exit status, receiving a kernel-generated signal, or receiving a SIGTERM or SIGKILL signal that is not sent because of user request.
term	(Mandatory) Normal process termination.
start	(Mandatory) Process start.
job_id	(Optional) Number assigned to the EEM policy that published the process event. Number is set to 798, because all other numbers are reserved for Cisco use.
instance	(Optional) Process instance ID. If specified, this argument must be an integer between 1 and 4294967295, inclusive.
path	(Optional) Process pathname (regular expression string).

node	(Optional) The node name is a string that consists of the word "node" followed by two fields separated by a slash (/), using the following format: node<slot-number>/<cpu-number> The slot-number is the hardware slot number. The cpu-number is the hardware CPU number. For example, the SP CPU in a Supervisor card on a Cisco Catalyst 6500 series switch located in slot 0 would be specified as node0/0. The RP CPU in a Supervisor card on a Cisco Catalyst 6500 series switch located in slot 0 would be addressed as node0/1. If the *node* argument is not specified, the default node specification is always the regular expression pattern match of * representing all applicable nodes.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.
tag	Tag is acceptable but ignored. Cisco IOS EEM scripts with the tag option can run in an Cisco IOS XR software environment without any error. Since Cisco IOS XR software does not support multiple events, the tag has no effect.

If an optional argument is not specified, the event matches all possible values of the argument. If multiple arguments are specified, the process event will be raised when all the conditions are matched.

Result String

None

Set _cerrno

No

event_register_snmp

Registers for a Simple Network Management Protocol (SNMP) statistics event. Use this Tcl command extension to run a policy when a given counter specified by an SNMP object ID (oid) crosses a defined threshold. When a snmp policy is registered, a poll timer is specified. Event matching occurs when the poll timer for the registered event expires. The **snmp-server manager** CLI command must be enabled for the SNMP notifications to work using Tcl policies.

Syntax

```
event_register_snmp [tag ?] oid ? get_type exact|next
entry_op gt|ge|eq|ne|lt|le entry_val ?
entry_type value|increment|rate
[exit_comb or|and]
[exit_op gt|ge|eq|ne|lt|le] [exit_val ?]
[exit_type value|increment|rate]
```

```
[exit_time ?] poll_interval ? [average_factor ?]
[queue_priority low|normal|high|last]
[maxrun ?] [nice 0|1]
```

Aruguments

tag	(Optional) String identifying a tag that can be used with the trigger Tcl command extension to support multiple event statements within a Tcl script.
entry_op	(Mandatory) Entry comparison operator used to compare the current OID data value with the entry value; if true, an event will be raised and event monitoring will be disabled until exit criteria are met.
get_type	(Mandatory) Type of SNMP get operation that needs to be applied to the OID specified. If the get_type argument is "exact," the value of the specified OID is retrieved; if the get_type argument is "next," the value of the lexicographical successor to the specified OID is retrieved.
entry_val	(Mandatory) Value with which the current oid data value should be compared to decide if the SNMP event should be raised.
entry-type	Specifies a type of operation to be applied to the object ID specified by the entry-val argument. Value is defined as the actual value of the entry-val argument. Increment uses the entry-val field as an incremental difference and the entry-val is compared with the difference between the current counter value and the value when the event was last triggered (or the first polled sample if this is a new event). A negative value checks the incremental difference for a counter that is decreasing. Rate is defined as the average rate of change over a period of time. The time period is the average-factor value multiplied by the poll-interval value. At each poll interval the difference between the current sample and the previous sample is taken and recorded as an absolute value. An average of the previous average-factor value samples is taken to be the rate of change.
exit_comb	(Optional) Exit combination operator used to indicate the combination of exit condition tests required to decide if the exit criteria are met so that the event monitoring can be reenabled. If it is "and," both exit value and exit time tests must be passed to meet the exit criteria. If it is "or," either exit value or exit time tests can be passed to meet the exit criteria When exit_comb is "and," exit_op, and exit_val (exit_time) must exist. When exit_comb is "or," (exit_op and exit_val) or (exit_time) must exist.
exit_op	(Optional) Exit comparison operator used to compare the current oid data value with the exit value; if true, event monitoring for this event will be reenabled.
exit_val	(Optional) Value with which the current oid data value should be compared to decide if the exit criteria are met.

exit-type	(Optional) Specifies a type of operation to be applied to the object ID specified by the exit-val argument. If not specified, the value is assumed. Value is defined as the actual value of the exit-val argument. Increment uses the exit-val field as an incremental difference and the exit-val is compared with the difference between the current counter value and the value when the event was last triggered (or the first polled sample if this is a new event). A negative value checks the incremental difference for a counter that is decreasing. Rate is defined as the average rate of change over a period of time. The time period is the average-factor value multiplied by the poll-interval value. At each poll interval the difference between the current sample and the previous sample is taken and recorded as an absolute value. An average of the previous average-factor value samples is taken to be the rate of change.
exit_time	(Optional) Number of POSIX timer units after an event is raised when event monitoring will be enabled again. Specified in SSSSSSSSSS[.MMM] format where SSSSSSSSSS must be an integer number representing seconds between 0 and 4294967295, inclusive. MMM represents milliseconds and must be an integer number between 0 and 999.
poll_interval	(Mandatory) Interval between consecutive polls in POSIX timer units. Currently the interval is forced to be at least 1 second (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999).
average-factor	(Optional) Number in the range from 1 to 64 used to calculate the period used for rate-based calculations. The average-factor value is multiplied by the poll-interval value to derive the period in milliseconds. The minimum average factor value is 1.

Result string

None

Set _cerrno

No

event_register_snmp_notification

Registers for a Simple Network Management Protocol (SNMP) notification trap event. Use this Tcl command extension to run a policy when an SNMP trap with the specified SNMP object ID (oid) is encountered on a specific interface or address. The **snmp-server manager** CLI command must be enabled for the SNMP notifications to work using Tcl policies.

Syntax

```
event_register_snmp_notification [tag ?] oid ? oid_val ?
op {gt|ge|eq|ne|lt|le}
[src_ip_address ?]
[dest_ip_address ?]
[queue_priority {normal|low|high|last}]
[maxrun ?]
[nice {0|1}]
[default ?]
```

```
[direction {incoming|outgoing}]
[msg_op {drop|send}]
```

Argument

tag	(Optional) String identifying a tag that can be used with the trigger Tcl command extension to support multiple event statements within a Tcl script.
oid	(Mandatory) OID number of the data element in SNMP dot notation (for example, 1.3.6.1.2.1.2.1.0). If the specified OID ends with a dot (.), then all OIDs that start with the OID number before the dot are matched. It supports all OID supported by SNMP in XR.
oid_val	(Mandatory) OID value with which the current OID data value should be compared to decide if the SNMP event should be raised.
op	(Mandatory) Comparison operator used to compare the current OID data value with the SNMP Protocol Data Unit (PDU) OID data value; if this is true, an event is raised.
src_ip_address	(Optional) Source IP address where the SNMP notification trap originates. The default is all; it is set to receive SNMP notification traps from all IP addresses. This option will not be supported in XR as src_ip_address is only for incoming trap which is not supported in EEM XR.
dest_ip_address	(Optional) Destination IP address where the SNMP notification trap is sent. The default is all; it is set to receive SNMP traps from all destination IP addresses.
default	(Optional) Specifies the time period in seconds during which the snmp notification event detector waits for the policy to exit. Thetime periodis specified in sssssssss[.mmm] format, where sssssssss must be an integer representing seconds between 0 and 4294967295 and mmm must be an integer representing milliseconds between 0 and 999
direction	(Optional) The direction of the incoming or outgoing SNMP trap or inform PDU to filter. The default value is outgoing. For XR direction incoming will not be supported and policy registration will fail if user provides direction as incoming.
msg_op	(Optional) The action to be taken on the SNMP PDU (drop it or send it) once the event is triggered. The default value is send. For XR msg_op drop will not be supported and policy registration will fail if user provides msg_op as drop.

Result String

None

Set _cerrno

No

event_register_stat

Registers for a statistics event. Use this Tcl command extension to run a policy when a given statistical counter crosses a defined threshold.

The following three fields are listed to uniquely identify the statistics counter that the EEM keyword monitors:

- Data element name corresponds to the argument name. For example, the ifstats-generic name is defined as interface generic statistics.
- The first modifier of the data element corresponds to the *modifier_1* argument. For example, Ethernet1_0 is defined as the first modifier for ifstats-generic, which qualifies the interface generic statistics to be specific for the Ethernet interface.
- The second modifier of the data element corresponds to the *modifier_2* argument. For example, input-ptks is defined as the second modifier for ifstats-generic, which further qualifies the interface statistics for the specific Ethernet interface is the number of packets received.

Syntax

```
event_register_stat name ? [modifier_1 ?] [modifier_2 ?]

entry_op gt|ge|eq|ne|lt|le entry_val ? [exit_comb or|and]

[exit_op gt|ge|eq|ne|lt|le] [exit_val ?] [exit_time_sec ?] [exit_time_nsec ?]
[poll_interval_sec ?] [poll_interval_nsec ?] [priority normal|low|high]
[maxrun_sec ?] [maxrun_nsec ?] [nice 0|1] [tag ?]
```

Arguments

name	(Mandatory) Statistics data element name.
modifier_1	Mandatory for interface statistics but optional for others. For interface statistics, this variable is the interface name. To get the interface name, use the **show interface brief** command. This command lists all the currently configured interface names designated by a slash (/), for example, Ethernet 1/0. When you want this interface to be configured for the *modifier_1* argument, change the slash to an underscore.
modifier_2	Mandatory for interface statistics but optional for others. For interface statistics, this variable is the interface statistic name. To get the interface statistic name, use the **show event manager statistics -table** command with the **all** keyword to list all the classes of statistics. Then, use the **show event manager statistics -table** command with the *name* argument to get the specific statistics name for modifier_2.
entry_op	(Mandatory) Entry comparison operator that is used to compare the current statistics value with the entry value. If true, an event is raised and event monitoring is disabled until the exit criteria is met.
entry_val	(Mandatory) Value in which the current statistical counter value that is compared to decide if the statistical event can be raised.
exit_comb	(Mandatory) Exit combination operator that indicates the combination of exit condition tests that are required to decide if the exit criteria is met so that event monitoring is reenabled. If so, both exit value and exit time tests must be passed to meet the exit criteria.Or either exit value or exit time tests are passed to meet the exit criteria. *exit_comb* and *exit_op*, *exit_val* arguments (*exit_time_sec* argument or *exit_time_nsec* argument) must exist. *exit_comb* argument or (*exit_op* and *exit_val* arguments) or (*exit_time_sec* argument or *exit_time_nsec* argument) must exist.

exit_op	Exit comparison operator that is used to compare the current statistics value with the exit value. If true, event monitoring for this event is reenabled.
exit_val	Value in which the current statistical counter value is compared to decide if the exit criteria is met.
exit_time_sec exit_time_nse	Number of POSIX timer units after the event is raised when event monitoring is enabled again. The integer number must be between 0 and 2147483647, inclusive.
poll_interval_sec poll_interval_nsec	Either the *poll_interval_sec* or *poll_interval_nsec* arguments must be specified. The interval must be between the consecutive polls in POSIX time units. Currently, it is forced to be at least one second. The integer number must be between 0 and 2147483647, inclusive.
priority	(Optional) Priority level that is queued for the script. If not specified, the default is using the normal priority.
maxrun_sec, maxrun_nsec	(Optional) Maximum run time of the script that is specified in seconds and nanoseconds. If not specified, 20-second run-time limit is used as the default. The integer number must be between 0 and 2147483647, inclusive.
nice	(Optional) When the *nice* argument is set to the value of 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.
tag	Tag is acceptable but ignored.Cisco IOS EEM scripts with the tag option can run in an Cisco IOS XR software environment without any error. Since Cisco IOS XR software does not support multiple events, the tag has no effect.

Note Exit criteria can be time-based, value-based, or both. Event monitoring is not reenabled until the exit criteria is met.

If multiple conditions exist, the statistics event is raised when all of the conditions are satisfied.

Reslt String

None

Set _cerrno

No

event_register_syslog

Registers for a syslog event. Use this Tcl command extension to trigger a policy when a syslog message of a specific pattern is logged after a certain number of occurrences during a certain period of time.

Syntax

```
event_register_syslog [occurs ?] [period ?] pattern ?
[priority all|emergencies|alerts|critical|errors|warnings|notifications|
```

```
informational|debugging|0|1|2|3|4|5|6|7]
[queue_priority low|normal|high]
[severity_fatal] [severity_critical] [severity_major]
[severity_minor] [severity_warning] [severity_notification]
[severity_normal] [severity_debugging]
[maxrun ?] [nice 0|1]
```

Arguments

occurs	(Optional) Number of occurrences before the event is raised; if not specified, the event is raised on the first occurrence. If specified, the value must be greater than 0.
period	(Optional) Time interval, in seconds and milliseconds, during which the one or more occurrences must take place in order to raise an event (specified in SSSSSSSSSS[.MMM] format where SSSSSSSSSS must be an integer number representing seconds between 0 and 4294967295, inclusive, and where MMM represents milliseconds and must be an integer number between 0 and 999). If this argument is not specified, no period check is applied.
pattern	(Mandatory) Regular expression used to perform syslog message pattern match. This argument is what the policy uses to identify the logged syslog message.
priority	(Optional) Message priority to be screened. If this argument is specified, only messages that are at the specified logging priority level, or lower, are screened. If this argument is not specified, the default priority is 0.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

If multiple conditions are specified, the syslog event is raised when all the conditions are matched.

Table 15: Severity Level Mapping For Syslog Events

Severity Keyword	Syslog Priority	Description
severity_fatal	LOG_EMERG (0)	System is unusable.
severity_critical	LOG_ALERT (1)	Critical conditions, immediate attention required.
severity_major	LOG_CRIT (2)	Major conditions.
severity_minor	LOG_ERR (3)	Minor conditions.
severity_warning	LOG_WARNING (4)	Warning conditions.
severity_notification	LOG_NOTICE (5)	Basic notification, informational messages.

Severity Keyword	Syslog Priority	Description
severity_normal	LOG_INFO (6)	Normal event, indicates returning to a normal state.
severity_debugging	LOG_DEBUG (7)	Debugging messages.

Result String

None

Set _cerrno

No

event_register_timer

Creates a timer and registers for a timer event as both a publisher and a subscriber. Use this Tcl command extension when there is a need to trigger a policy that is time specific or timer based. This event timer is both an event publisher and a subscriber. The publisher part indicates the conditions under which the named timer is to go off. The subscriber part identifies the name of the timer to which the event is subscribing.

Note Both the CRON and absolute time specifications work on local time.

Syntax

```
event_register_timer watchdog|countdown|absolute|cron
[name ?] [cron_entry ?]
[time ?]
[queue_priority low|normal|high] [maxrun ?]
[nice 0|1]
```

Arguments

watchdog	(Mandatory) Watchdog timer.
countdown	(Mandatory) Countdown timer.
absolute	(Mandatory) Absolute timer.
cron	(Mandatory) CRON timer.
name	(Optional) Name of the timer.

cron_entry	(Optional) Entry must be specified if the CRON timer type is specified. Must not be specified if any other timer type is specified. A cron_entry is a partial UNIX crontab entry (the first five fields) as used with the UNIX CRON daemon. A cron_entry specification consists of a text string with five fields. The fields are separated by spaces. The fields represent the time and date when CRON timer events will be triggered. The fields are described in Table 16: Time and Date When CRON Events Will Be Triggered , on page 71 . Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an hour entry specifies execution at hours 8, 9, 10, and 11. A field may be an asterisk (*), which always stands for "first-last." Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: "1,2,5,9" and "0-4,8-12". Step values can be used in conjunction with ranges. Following a range with "/<number>" specifies skips of the number's value through the range. For example, "0-23/2" is used in the hour field to specify an event that is triggered every other hour. Steps are also permitted after an asterisk, so if you want to say "every two hours", use "*/2". Names can also be used for the month and the day of week fields. Use the first three letters of the particular day or month (case does not matter). Ranges or lists of names are not allowed. The day on which a timer event is triggered can be specified by two fields: day of month and day of week. If both fields are restricted (that is, are not *), an event will be triggered when either field matches the current time. For example, "30 4 1,15 * 5" would cause an event to be triggered at 4:30 a.m. on the 1st and 15th of each month, plus every Friday. Instead of the first five fields, one of seven special strings may appear. These seven special strings are described in Table 17: Special Strings for cron_entry, on page 71. Example 1: "0 0 1,15 * 1" would trigger an event at midnight on the 1st and 15th of each month, as well as on every Monday. To specify days by only one field, the other field should be set to *; "0 0 * * 1" would trigger an event at midnight only on Mondays. Example 2: "15 16 1 * *" would trigger an event at 4:15 p.m. on the first day of each month. Example 3: "0 12 * * 1-5" would trigger an event at noon on Monday through Friday of each week. Example 4: "@weekly" would trigger an event at midnight once a week on Sunday.
time	(Optional) Time must be specified if a timer type other than CRON is specified. Must not be specified if the CRON timer type is specified. For watchdog and countdown timers, the number of seconds and milliseconds until the timer expires; for the absolute timer, the calendar time of the expiration time. Time is specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999. An absolute expiration date is the number of seconds and milliseconds since January 1, 1970. If the date specified has already passed, the timer expires immediately.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.

maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Table 16: Time and Date When CRON Events Will Be Triggered

Field	Allowed Values
minute	0-59
hour	0-23
day of month	1-31
month	1-12 (or names, see Table 17: Special Strings for cron_entry, on page 71)
day of week	0-7 (0 or 7 is Sun, or names; see Table 17: Special Strings for cron_entry, on page 71)

Table 17: Special Strings for cron_entry

String	Meaning
@yearly	Trigger once a year, "0 0 1 1 *".
@annually	Same as @yearly.
@monthly	Trigger once a month, "0 0 1 * *".
@weekly	Trigger once a week, "0 0 * * 0".
@daily	Trigger once a day, "0 0 * * *".
@midnight	Same as @daily.
@hourly	Trigger once an hour, "0 * * * *".

Result String

None

Set _cerrno

No

See Also

event_register_timer_subscriber, on page 72

event_register_timer_subscriber

Registers for a timer event as a subscriber. Use this Tcl command extension to identify the name of the timer to which the event timer, as a subscriber, wants to subscribe. The event timer depends on another policy or another process to actually manipulate the timer. For example, let policyB act as a timer subscriber policy, but policyA (although it does not need to be a timer policy) uses register_timer, timer_arm, or timer_cancel Tcl command extensions to manipulate the timer referenced in policyB.

Syntax

```
event_register_timer_subscriber watchdog|countdown|absolute|cron
name ? [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

Arguments

watchdog	(Mandatory) Watchdog timer.
countdown	(Mandatory) Countdown timer.
absolute	(Mandatory) Absolute timer.
cron	(Mandatory) CRON timer.
name	(Mandatory) Name of the timer.
queue_priority	(Optional) Priority level at which the script will be queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the nice argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Note An EEM policy that registers for a timer event or a counter event can act as both publisher and subscriber.

Result String

None

Set _cerrno

No

See Also

event_register_timer, on page 69

event_register_track

Registers for a report event from the Object Tracking component in XR. Use this Tcl command extension to trigger a policy on the basis of a Object Tracking component report for a specified track. This will be implemented as a new process in IOS-XR which will be dlrsc_tracker. Please note that the manageability package should be installed for the track ED to be functional.

Syntax

```
event_register_track ? [tag ?] [state up|down|any] [queue_priority low|normal|high|last]
[maxrun ?]
[nice 0|1]
```

Arguments

? (represents a string)	(Mandatory) Tracked object name.
tag	(Optional) String identifying a tag that can be used with the trigger Tcl command extension to support multiple event statements within a Tcl script.
state	(Optional) Specifies that the tracked object transition will cause an event to be raised. If **up** is specified, an event will be raised when the tracked object transitions from a down state to an up state. If **down** is specified, an event will be raised when the tracked object transitions from an up state to a down state. If **any** is specified, an event will be raised when the tracked object transitions to or from any state.
queue_priority	(Optional) Priority level at which the script will be queued: • queue_priority low-Specifies that the script is to be queued at the lowest of the three priority levels. • queue_priority normal-Specifies that the script is to be queued at a priority level greater than low priority but less than high priority. • queue_priority high-Specifies that the script is to be queued at the highest of the three priority levels. • queue_priority last-Specifies that the script is to be queued at the lowest priority level. If more than one script is registered with the "queue_priority_last" argument set, these scripts will execute in the order in which the events are published. **Note** The queue_priority argument specifies the queuing priority, but not the execution priority, of the script being registered. If this argument is not specified, the default queuing priority is normal.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the nice argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

If an optional argument is not specified, the event matches all possible values of the argument.

Result String

None

Set _cerrno

No

event_register_wdsysmon

Registers for a Watchdog system monitor event. Use this Tcl command extension to register for a composite event which is a combination of several subevents or conditions. For example, you can use the **event_register_wdsysmon** command to register for the combination of conditions wherein the CPU usage of a certain process is over 80 percent, and the memory used by the process is greater than 50 percent of its initial allocation. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
event_register_wdsysmon [timewin ?]
[sub12_op and|or|andnot]
[sub23_op and|or|andnot]
[sub34_op and|or|andnot]
[sub1 subevent-description]
[sub2 subevent-description]
[sub3 subevent-description]
[sub4 subevent-description] [node ?]
[queue_priority low|normal|high]
[maxrun ?] [nice 0|1]
```

Arguments

timewin	(Optional) Time window within which all of the subevents have to occur in order for an event to be generated and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999).
sub12_op	(Optional) Combination operator for comparison between subevent 1 and subevent 2.
sub34_op	(Optional) Combination operator for comparison between subevent 1 and 2, subevent 3, and subevent 4.
sub1	(Optional) Subevent 1 is specified.
subevent-description	(Optional) Syntax for the subevent.
sub2	(Optional) Subevent 2 is specified.
sub3	(Optional) Subevent 3 is specified.
sub4	(Optional) Subevent 4 is specified.

node	(Optional) Node name to be monitored for deadlock conditions is a string that consists of the word 'node', which is followed by two fields separated by a slash (/) using the following format: `node<slot-number>/<cpu-number>` The slot-number is the hardware slot number. The cpu-number is the hardware CPU number. For example, the SP CPU in a Supervisor card on a Cisco Catalyst 6500 Series Switch located in slot 0 is specified as node0/0. The RP CPU in a Supervisor card on a Cisco Catalyst 6500 Series Switch located in slot 0 is addressed as node0/1. If the node argument is not specified, the default node specification is the local node on which the registration is done.
queue_priority	(Optional) Priority level at which the script is queued; normal priority is greater than low priority but less than high priority. The priority here is not execution priority, but queuing priority. If this argument is not specified, the default priority is normal.
maxrun	(Optional) Maximum run time of the script that is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999. If this argument is not specified, the default 20-second run-time limit is used.
nice	(Optional) Policy run-time priority setting. When the *nice* argument is set to 1, the policy is run at a run-time priority that is less than the default priority. The default value is 0.

Subevents

The syntax of subevent descriptions can be one of seven cases.

For arguments in subevent description, the following constraints apply on the value of number arguments:

- For dispatch_mgr, val must be an integer between 0 and 4294967295, inclusive.
- For cpu_proc and cpu_tot, val must be an integer between 0 and 100, inclusive.
- For mem_proc, mem_tot_avail, and mem_tot_used, if is_percent is FALSE, val must be an integer between 0 and 4294967295, inclusive.

1. deadlock procname ?

Arguments

procname	(Mandatory) Regular expression that specifies the process name that you want to monitor for deadlock conditions. This subevent ignores the time window even if it is given.

1. dispatch_mgr [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

Arguments

procname	(Optional) Regular expression that specifies the process name that you want to monitor for the dispatch_manager status.
op	(Optional) Comparison operator that is used to compare the collected number of events with the specified value. If true, an event is raised.
val	(Optional) Value in which the number of events that have occurred is compared.
period	(Optional) Time period for the number of events that have occurred and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999. If this argument is not specified, the most recent sample is used.

1. cpu_proc [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

Arguments

procname	(Optional) Regular expression that specifies the process name that you want to monitor for CPU utilization conditions.
op	(Optional) Comparison operator that is used to compare the collected CPU usage sample percentage with the specified percentage value. If true, an event is raised.
val	(Optional) Percentage value in which the average CPU usage during the sample period is compared.
period	(Optional) Time period for averaging the collection of samples and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999. If this argument is not specified, the most recent sample is used.

1. cpu_tot [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

Arguments

op	(Optional) Comparison operator that is used to compare the collected total system CPU usage sample percentage with the specified percentage value. If true, an event is raised.
val	(Optional) Percentage value in which the average CPU usage during the sample period is compared.
period	(Optional) Time period for averaging the collection of samples and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999. If this argument is not specified, the most recent sample is used.

1. mem_proc [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

Arguments

procname	(Optional) Regular expression that specifies the process name that you want to monitor for memory usage.
op	(Optional) Comparison operator that is used to compare the collected memory used with the specified value. If true, an event is raised.
val	(Optional) Percentage or an absolute value that is specified in kilobytes. A percentage represents the difference between the oldest sample in the specified time period and the latest sample. If memory usage increased from 150 KB to 300 KB within the time period, the percentage increase is 100. This is the value in which the measured value is compared.
is_percent	(Optional) If set to TRUE, the percentage value is collected and compared. Otherwise, the absolute value is collected and compared.
period	(Optional) If is_percent is set to TRUE, the time period for the percentage is computed. Otherwise, the time period for the collection samples is averaged and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999. If this argument is not specified, the most recent sample is used.

1. mem_tot_avail [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

Arguments

op	(Optional) Comparison operator that is used to compare the collected available memory with the specified value. If true, an event is raised.
val	(Optional) Percentage or an absolute value that is specified in kilobytes. A percentage represents the difference between the oldest sample in the specified time period and the latest sample. If available memory usage has decreased from 300 KB to 150 KB within the time period, the percentage decrease is 50. This is the value in which the measured value is compared.
is_percent	(Optional) If set to TRUE, the percentage value is collected and compared. Otherwise, the absolute value is collected and compared.
period	(Optional) If is_percent is set to TRUE, the time period for the percentage is computed. Otherwise, the time period for the collection samples is averaged and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the most recent sample is used.

1. mem_tot_used [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

Arguments

op	(Optional) Comparison operator that is used to compare the collected used memory with the specified value. If true, an event is raised.

val	(Optional) Percentage or an absolute value that is specified in kilobytes. A percentage represents the difference between the oldest sample in the specified time period and the latest sample. If memory usage has increased from 150 KB to 300 KB within the time period, the percentage increase is 100. This is the value in which the measured value is compared.
is_percent	(Optional) If set to TRUE, the percentage value is collected and compared. Otherwise, the absolute value is collected and compared.
period	(Optional) If is_percent is set to TRUE, the time period for the percentage is computed. Otherwise, the time period for the collection samples is averaged and is specified in SSSSSSSSSS[.MMM] format. SSSSSSSSSS format must be an integer representing seconds between 0 and 4294967295, inclusive. MMM format must be an integer representing milliseconds between 0 and 999). If this argument is not specified, the most recent sample is used. **Note** This argument is mandatory if is_percent is set to TRUE; otherwise, it is optional.

Result String

None

Set _cerrno

No

Note Inside a subevent description, each argument is position as independent.

Embedded Event Manager Event Information Tcl Command Extension

The following EEM Event Information Tcl Command Extensions are supported:

event_reqinfo

Queries information for the event that caused the current policy to run.

Syntax

```
event_reqinfo
```

Arguments

None

Result String

If the policy runs successfully, the characteristics for the event that triggered the policy will be returned. The following sections show the characteristics returned for each event detector.

For EEM_EVENT_APPLICATION

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"sub_system 0x%x type %u data1 {%s} data2 {%s} data3 {%s} data4 {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	The time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
sub_system	Number assigned to the EEM policy that published the application event. Number is set to 798 because all other numbers are reserved for Cisco use.
type	Event subtype within the specified component.
data1data2data3data4	Argument data that is passed to the application-specific event when the event is published. The data is character text, an environment variable, or a combination of the two.

For EEM_EVENT_COUNTER

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"name {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	The time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
name	Counter name.

For EEM_EVENT_NONE

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.

For EEM_EVENT_OIR

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"slot %u event %s"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event ID.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
slot	Slot number for the affected card.
event	Indicates a string, removed or online, that represents either an OIR removal event or an OIR insertion event.

For EEM_EVENT_PROCESS (Software Modularity Only)

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"sub_system 0x%x instance %u process_name {%s} path {%s} exit_status 0x%x"
"respawn_count %u last_respawn_sec %ld last_respawn_msec %ld fail_count %u"
"dump_count %u node_name {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.

Event Type	Description
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
sub_system	Number assigned to the EEM policy that published the application-specific event. Number is set to 798 because all other numbers are reserved for Cisco use.
instance	Process instance ID.
process_name	Process name.
path	Process absolute name including path.
exit_status	Process last exit status.
respawn_count	Number of times that the process was restarted.
last_respawn_seclast_respawn_msec	Calendar time when the last restart occurred.
fail_count	Number of restart attempts of the process that failed. This count will be reset to 0 when the process is successfully restarted.
dump_count	Number of core dumps taken of the process.
node_name	Name of the node that the process is on. The node name is a string that consists of the word “node” followed by two fields separated by a slash character using the following format: node<slot-number>/<cpu-number> The slot-number is the hardware slot number. The cpu-number is the hardware CPU number.

For EEM_EVENT_RF

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"event {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.

Event Type	Description
event	RF progression or status event notification that caused this event to be published.

For EEM_EVENT_SYSLOG_MSG

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"msg {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
msg	Last syslog message that matches the pattern.

For EEM_EVENT_TIMER_ABSOLUTE

EEM_EVENT_TIMER_COUNTDOWN

EEM_EVENT_TIMER_WATCHDOG

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"timer_type %s timer_time_sec %ld timer_time_msec %ld"
"timer_remain_sec %ld timer_remain_msec %ld"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.

Event Type	Description
timer_type	Type of the timer. Can be one of the following: • watchdog • countdown • absolute
timer_time_sectimer_time_msec	Time when the timer expired.
timer_remain_sectimer_remain_msec	Remaining time before the next expiration.

For EEM_EVENT_TIMER_CRON

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"timer_type {%s} timer_time_sec %ld timer_time_msec %ld"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
timer_type	Type of the timer.
timer_time_sectimer_time_msec	Time when the timer expired.

For EEM_EVENT_TRACK

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"track_number {%u} track_state {%s}"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event ID.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.

Event Type	Description
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
track_number	Number of the tracked object that caused the event to be triggered.
track_state	State of the tracked object when the event was triggered; valid states are up or down.

For EEM_EVENT_WDSYSMON

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"num_subs %u"
```

Event Type	Description
event_id	Unique number that indicates the ID for this published event. Multiple policies may be run for the same event, and each policy will have the same event_id.
event_type	Type of event.
event_type_string	ASCII string that represents the name of the event for this event type.
event_pub_secevent_pub_msec	Time, in seconds and milliseconds, when the event was published to the Embedded Event Manager.
num_subs	Subevent number.

Where the subevent info string is for a deadlock subevent:

```
"{type %s num_entries %u entries {entry 1, entry 2, ...}}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
num_entries	Number of processes and threads in the deadlock.
entries	Information of processes and threads in the deadlock.

Where each entry is:

```
"{node {%s} procname {%s} pid %u tid %u state %s b_node %s b_procname %s b_pid %u
b_tid %u}"
```

Assume that the entry describes the scenario in which Process A thread m is blocked on process B thread n:

Subevent Type	Description
node	Name of the node that process A thread m is on.
procname	Name of process A.
pid	Process ID of process A.
tid	Thread ID of process A thread m.
state	Thread state of process A thread m. Can be one of the following: • STATE_CONDVAR • STATE_DEAD • STATE_INTR • STATE_JOIN • STATE_MUTEX • STATE_NANOSLEEP • STATE_READY • STATE_RECEIVE • STATE_REPLY • STATE_RUNNING • STATE_SEM • STATE_SEND • STATE_SIGSUSPEND • STATE_SIGWAITINFO • STATE_STACK • STATE_STOPPED • STATE_WAITPAGE • STATE_WAITTHREAD
b_node	Name of the node that process B thread is on.
b_procname	Name of process B.
b_pid	Process ID of process B.
b_tid	Thread ID of process B thread n; 0 means that process A thread m is blocked on all threads of process B.

For dispatch_mgr Subevent

```
"{type %s node {%s} procname {%s} pid %u value %u sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node that the POSIX process is on.
procname	POSIX process name for this subevent.
pid	POSIX process ID for this subevent. **Note** The three preceding fields describe the owner process of this dispatch manager.
value	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the number of events processed by the dispatch manager is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the total number of events processed by this dispatch manager is in the given time window.
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, the sec and msec variables are the actual time difference between the time stamps of the oldest and latest samples in this time window.

For cpu_proc Subevent

```
"{type %s node {%s} procname {%s} pid %u value %u sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node that the POSIX process is on.
procname	POSIX process name for this subevent.
pid	POSIX process ID for this subevent. **Note** The three preceding fields describe the process whose CPU utilization is being monitored.
value	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the process CPU utilization is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the averaged process CPU utilization is in the given time window.

Subevent Type	Description
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, the sec and msec variables are the actual time difference between the time stamps of the oldest and latest samples in this time window.

For cpu_tot Subevent

```
"{type %s node {%s} value %u sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node on which the total CPU utilization is being monitored.
value	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the total CPU utilization is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the averaged total CPU utilization is in the given time window.
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, the sec and msec variables are the actual time difference between the time stamps of the oldest and latest samples in this time window.

For mem_proc Subevent

```
"{type %s node {%s} procname {%s} pid %u is_percent %s value %u diff %d sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node that the POSIX process is on.
procname	POSIX process name for this subevent.
pid	POSIX process ID for this subevent. **Note** The three preceding fields describe the process whose memory usage is being monitored.

Subevent Type	Description
is_percent	Can be either TRUE or FALSE. TRUE means that the value is a percentage value; FALSE means that the value is an absolute value (may be an averaged value).
value	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the process used memory is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the averaged process used memory utilization is in the given time window.
diff	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the diff is the percentage difference between the first process used memory sample ever collected and the latest process used memory sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the diff is the percentage difference between the oldest and latest process used memory utilization in the specified time window.
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, the sec and msec variables are the actual time difference between the time stamps of the oldest and latest samples in this time window.

If the *is_percent* argument is FALSE, and the *sec* and *msec* arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *value* is the process used memory in the latest sample.
- *diff* is 0.
- *sec* and *msec* are both 0.

If the *is_percent* argument is FALSE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- *value* is the averaged process used memory sample value in the specified time window.
- *diff* is 0.
- *sec* and *msec* are both the actual time difference between the time stamps of the oldest and latest samples in this time window.

If the *is_percent* argument is TRUE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- value is 0.
- *diff* is the percentage difference between the oldest and latest process used memory samples in the specified time window.
- *sec* and **msec** are the actual time difference between the time stamps of the oldest and latest process used memory samples in this time window.

If the *is_percent* argument is TRUE, and the *sec* and *msec* arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *value* is 0.
- *diff* is the percentage difference between the first process used memory sample ever collected and the latest process used memory sample.
- *sec* and *msec* are the actual time difference between the time stamps of the first process used memory sample ever collected and the latest process used memory sample.

For mem_tot_avail Subevent

```
"{type %s node {%s} is_percent %s used %u avail %u diff %d sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node for which the total available memory is being monitored.
is_percent	Can be either TRUE or FALSE. TRUE means that the value is a percentage value; FALSE means that the value is an absolute value (may be an averaged value).
used	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the total used memory is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the averaged total used memory utilization is in the given time window.
avail	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the avail is in the latest total available memory sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the avail is the total available memory utilization in the specified time window.
diff	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the diff is the percentage difference between the first total available memory sample ever collected and the latest total available memory sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the diff is the percentage difference between the oldest and latest total available memory utilization in the specified time window.
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, they are the actual time difference between the time stamps of the oldest and latest samples in this time window.

If the *is_percent* argument is FALSE, and the sec and msec arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *used* is the total used memory in the latest sample.
- *avail* is the total available memory in the latest sample.
- *diff* is 0.
- *sec* and **msec** are both 0.

If the *is_percent* argument is FALSE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- *used* is 0.
- *avail* is the averaged total available memory sample value in the specified time window.
- *diff* is 0.
- *sec* and *msec* are both the actual time difference between the time stamps of the oldest and latest total available memory samples in this time window.

If the *is_percent* argument is TRUE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- *used* is 0.
- *avail* is 0.
- *diff* is the percentage difference between the oldest and latest total available memory samples in the specified time window.
- *sec* and *msec* are both the actual time difference between the time stamps of the oldest and latest total available memory samples in this time window.

If the *is_percent* argument is TRUE, and the *sec* and *msec* arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *used* is 0.
- *avail* is 0.
- *diff* is the percentage difference between the first total available memory sample ever collected and the latest total available memory sample.
- *sec* and msec are the actual time difference between the time stamps of the first total available memory sample ever collected and the latest total available memory sample.

For mem_tot_used Subevent

```
"{type %s node {%s} is_percent %s used %u avail %u diff %d sec %ld msec %ld}"
```

Subevent Type	Description
type	Type of wdsysmon subevent.
node	Name of the node for which the total used memory is being monitored.
is_percent	Can be either TRUE or FALSE. TRUE means that the value is a percentage value; FALSE means that the value is an absolute value (may be an averaged value).

Subevent Type	Description
used	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the total used memory is in the latest sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the averaged total used memory utilization is in the given time window.
avail	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the avail is in the latest total used memory sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the avail is the total used memory utilization in the specified time window.
diff	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, the diff is the percentage difference between the first total used memory sample ever collected and the latest total used memory sample. If a time window is specified and is greater than zero in the event registration Tcl command extension, the diff is the percentage difference between the oldest and latest total used memory utilization in the specified time window.
secmsec	If the sec and msec variables are specified as 0 or are unspecified in the event registration Tcl command extension, they are both 0. If a time window is specified and is greater than zero in the event registration Tcl command extension, the sec and msec variables are the actual time difference between the time stamps of the oldest and latest samples in this time window.

If the *is_percent* argument is FALSE, and the *sec* and *msec* arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *used* is the total used memory in the latest sample,
- *avail* is the total available memory in the latest sample,
- *diff* is 0,
- *sec* and *msec* are both 0,

If the *is_percent* argument is FALSE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- *used* is the averaged total used memory sample value in the specified time window,
- *avail* is 0,
- *diff* is 0,
- *sec* and *msec* are both the actual time difference between the time stamps of the oldest and latest total used memory samples in this time window,

If the *is_percent* argument is TRUE, and a time window is specified as greater than zero in the event registration Tcl command extension:

- *used* is 0.
- *avail* is 0.

- *diff* is the percentage difference between the oldest and latest total used memory samples in the specified time window.
- *sec* and *msec* are both the actual time difference between the time stamps of the oldest and latest total used memory samples in this time window.

If the *is_percent* argument is TRUE, and the sec and msec arguments are specified as 0 or are unspecified in the event registration Tcl command extension:

- *used* is 0.
- *avail* is 0.
- *diff* is the percentage difference between the first total used memory sample ever collected and the latest total used memory sample.
- *sec* and *msec* are the actual time difference between the time stamps of the first total used memory sample ever collected and the latest total used memory sample.

Set _cerrno

Yes

event_reqinfo_multi

Adds a new function to retrieve the event_reqinfo data for every event that contributed to the triggering of the script. The data returned will be a list of result strings indexed by event specification tag. Error processing is the same as in event_reqinfo function.

Syntax

```
event_reqinfo_multi
```

Arguments

None

Result String

The following section shows the result string from the event reqinfo multi call:

```
"<ev-tag> {event_id %u event_type %u event_type_string
{%s} event_pub_sec %ld event_pub_msec %ld timer_type {%s} timer_time_sec
%ld timer_time_msec %ld timer_remain_sec %ld timer_remain_msec %ld}
                 <ev-tag> {event_id %u event_type %u event_type_string
{%s} event_pub_sec %ld event_pub_msec %ld oid {%s} val {%s} delta_val
{%s} exit_event {%s}}"
               Typical usage for a multi-event consisting of both a timer event and an
SNMP event might be:
               array set arr_minfo [event_reqinfo_multi]
               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
               }
               array set arr_einfo $arr_minfo(<ev-tag-for-timer-event-spec>)
               global timer_type timer_time_sec
```

```
set timer_type $arr_einfo(timer_type)
set timer_time_sec $arr_einfo(timer_time_sec)
```

The output of event_reqinfo_multi is ordered from most recent to least recent event that contributed to the triggering of the policy.

Embedded Event Manager Event Publish Tcl Command Extension

event_publish appl

Publishes an application-specific event.

Syntax

```
event_publish sub_system ? type ? [arg1 ?] [arg2 ?] [arg3 ?] [arg4 ?]
```

Arguments

sub_system	(Mandatory) Number assigned to the EEM policy that published the application-specific event. Number is set to 798 because all other numbers are reserved for Cisco use.
type	(Mandatory) Event subtype within the specified component. The sub_system and type arguments uniquely identify an application event. Must be an integer between 1 and 4294967295, inclusive.
[arg1 ?]-[arg4 ?]	(Optional) Four pieces of application event publisher string data.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

Sample Usage

This example demonstrates how to use the **event_publish appl** Tcl command extension to execute a script *n* times repeatedly to perform some function (for example, to measure the amount of CPU time taken by a given group of Tcl statements). This example uses two Tcl scripts.

Script1 publishes a type 9999 EEM event to cause Script2 to run for the first time. Script1 is registered as a none event and is run using the Cisco IOS XR software CLI **event manager run** command. Script2 is registered as an EEM application event of type 9999, and this script checks to see if the application publish arg1 data (the iteration number) exceeds the EEM environment variable test_iterations value. If the test_iterations value is exceeded, the script writes a message and exits; otherwise the script executes the remaining statements and

reschedules another run. To measure the CPU utilization for Script2, use a value of test_iterations that is a multiple of 10 to calculate the amount of average CPU time used by Script2.

To run the Tcl scripts, enter the following Cisco IOS XR software commands:

```
configure terminal
 event manager environment test_iterations 100
 event manager policy script1.tcl
 event manager policy script2.tcl
 end
event manager run script1.tcl
```

The Tcl script Script2 is executed 100 times. If you execute the script without the extra processing and derive the average CPU utilization, and then add the extra processing and repeat the test, you can subtract the former CPU utilization from the later CPU utilization to determine the average for the extra processing.

Script1 (script1.tcl)

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

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
}

action_syslog priority info msg "EEM application_publish test start"
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
}

# Cause the first iteration to run.
event_publish sub_system 798 type 9999 arg1 0
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
}
```

Script2 (script2.tcl)

```
::cisco::eem::event_register_appl sub_system 798 type 9999

# Check if all the required environment variables exist.
# If any required environment variable does not exist, print out an error msg and quit.
if {![info exists test_iterations]} {
    set result \
      "Policy cannot be run: variable test_iterations has not been set"
    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
}
# Data1 contains the arg1 value used to publish this event.
set iter $arr_einfo(data1)

# Use the arg1 info from the previous run to determine when to end.
if {$iter >= $test_iterations} {
    # Log a message.
    action_syslog priority info msg "EEM application_publish test end"
    if {$_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
}
set iter [expr $iter + 1]

# Log a message.
set msg [format "EEM application_publish 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
}

# Do whatever processing that you want to measure here.

# Cause the next iteration to run. Note that the iteration is passed to the
# next operation as arg1.
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
}
```

Embedded Event Manager Multiple Event Support Tcl Command Extensions

Attribute

Specifies a complex event used for Multi Event Support.

Syntax

```
attribute tag ? [occurs ?]
```

Arguments

tag	Specifies a tag using the *event-tag* argument that can be used with the **attribute** command to associate an event.
occurs	(Optional) Specifies the number of occurrences before an EEM event is triggered. If not specified, an EEM event is triggered on the first occurrence. The range is from 1 to 4294967295

Result String

None

Example:

attribute tag 1 occurs 1

Correlate

Builds a single complex event and allows Boolean logic to relate events.

Syntax

```
correlate event ? event ?
```

Arguments

event	Specifies the event that can be used with the **trigger** command to support multiple event statements within an script. If the event associated with the *event-tag* argument occurs for the number of times specified by the **trigger** command, the result is true. If not, the result is false.
andnot	(Optional) Specifies that if event 1 occurs the action is executed, and if event 2 and event 3 occur together the action is not executed.
and	(Optional) Specifies that if event 1 occurs the action is executed, and if event 2 and event 3 occur together the action is executed.
or	(Optional) Specifies that if event 1 occurs the action is executed, or else if event 2 and event 3 occur together the action is executed.

Result String

None

Example:

correlate event 1 or event 2 and event 3

Trigger

Specifies the multiple event configuration ability of Embedded Event Manager (EEM) events. A multiple event is one that can involve one or more event occurrences and a time period for the event to occur. The events are raised based on the specified parameters.

Syntax

```
trigger [occurs ?] [period ?] [period-start ?] [delay ?]
```

Arguments

occurs	(Optional) Specifies the number of times the total correlation occurs before an EEM event is raised. When a number is not specified, an EEM event is raised on the first occurrence. The range is from 1 to 4294967295.
period	(Optional) Time interval in seconds and optional milliseconds, during which the one or more occurrences must take place. This is specified in the format sssssssssss[.mmm], where sssssssssss must be an integer number representing seconds between 0 and 4294967295, inclusive and mmm represents milliseconds and must be an integer number between 0 to 999.
period-start	(Optional) Specifies the start of an event correlation window. If not specified, event monitoring is enabled after the first CRON period occurs.
delay	(Optional) Specifies the number of seconds and optional milliseconds after which an event will be raised if all the conditions are true (specified in the format sssssssssss[.mmm], where sssssssssss must be an integer number representing seconds between 0 and 4294967295, inclusive and mmm represents milliseconds and must be an integer number between 0 to 999).

Result String

None

Example:

trigger occurs 1 period-start "0 8 * * 1-5" period 720

Embedded Event Manager Action Tcl Command Extensions

action_process

Starts, restarts, or shuts down a Software Modularity process. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
action_process start|restart|shutdown [job_id ?]
[process_name ?] [instance ?]
```

Arguments

start	(Mandatory) Specifies that a process is to be started.
restart	(Mandatory) Specifies that a process is to be restarted.
shutdown	(Mandatory) Specifies that a process is to be stopped (shut down).

job_id	(Optional) System manager assigned job ID for the process. If you specify this argument, it must be an integer between 1 and 4294967295, inclusive.
process_name	(Optional) Process name. Either job_id must be specified or process_name and instance must be specified.
instance	(Optional) Process instance ID. If you specify this argument, it must be an integer between 1 and 4294967295, inclusive.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

This error means that the action command requested was unknown.

```
(_cerr_sub_num = 425, _cerr_sub_err = 1) SYSMGR_ERROR_INVALID_ARGS  (Invalid arguments
passed)
```

This error means that the arguments passed in were invalid.

```
(_cerr_sub_num = 425, _cerr_sub_err = 2) SYSMGR_ERROR_NO_MEMORY  (Could not allocate required
 memory)
```

This error means that an internal SYSMGR request for memory failed.

```
(_cerr_sub_num = 425, _cerr_sub_err = 5) SYSMGR_ERROR_NO_MATCH  (This process is not known
 to sysmgr)
```

This error means that the process name was not known.

```
(_cerr_sub_num = 425, _cerr_sub_err = 14) SYSMGR_ERROR_TOO_BIG  (outside the valid limit)
```

This error means that an object size exceeded its maximum.

```
(_cerr_sub_num = 425, _cerr_sub_err = 15) SYSMGR_ERROR_INVALID_OP  (Invalid operation for
this process)
```

This error means that the operation was invalid for the process.

action_program

Allows a Tcl script to run a POSIX process (program), optionally with a given argument string, environment string, Standard Input (stdin) pathname, Standard Output (stdout) pathname, or Standard Error (stderr) pathname. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
action_program path ? [argv ?] [envp ?] [stdin ?] [stdout ?] [stderr ?]
```

Arguments

path	(Mandatory) Pathname of a program to run.
argv	(Optional) Argument string of the program.
envp	(Optional) Environment string of the program.
stdin	(Optional) Pathname for stdin.
stdout	(Optional) Pathname for stdout.
stderr	(Optional) Pathname for stderr.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

This error means that the action command requested was unknown.

```
(_cerr_sub_err = 34)    FH_EMAXLEN  (maximum length exceeded)
```

This error means that the object length or number exceeded the maximum.

action_script

Allows a Tcl script to enable or disable the execution of all Tcl scripts (enables or disables the script scheduler).

Syntax

```
action_script [status enable|disable]
```

Arguments

status	(Optional) Flag to indicate script execution status. If this argument is set to enable, script execution is enabled; if this argument is set to disable, script execution is disabled.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

This error means that the action command requested was unknown.

```
(_cerr_sub_err = 52)    FH_ECONFIG  (configuration error)
```

This error means that a configuration error has occurred.

action_setnode

Switches to the given node to enable subsequent EEM commands to be performed on that node. The following EEM commands use action_setnode to set their target node:

- action_process
- sys_reqinfo_proc
- sys_reqinfo_proc_all
- sys_reqinfo_crash_history
- sys_reqinfo_proc_version

Syntax

```
action_setnode [node ?]
```

Arguments

node	(Mandatory) Name of the node.

Result String

None

Set _cerrno

Yes

action_syslog

Logs a message.

Syntax

```
action_syslog [priority emerg|alert|crit|err|warning|notice|info|debug]
[msg ?]
```

Arguments

priority	(Optional) Action_syslog message facility level. If this argument is not specified, the default priority is LOG_INFO.
msg	(Optional) Message to be logged.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

This error means that the action command requested was unknown.

action_track_read

Reads the state of a tracked object when an Embedded Event Manager (EEM) script is triggered.

Syntax

```
action_track_read ?
```

Arguments

? (represents a string)	(Mandatory) Tracked object name.

Result String

```
name {%s}
state {%s}
```

Set _cerrno

Yes

```
FH_ENOTRACK
```

This error means that the tracked object name was not found.

Embedded Event Manager Utility Tcl Command Extensions

appl_read

Reads Embedded Event Manager (EEM) application volatile data. This Tcl command extension provides support for reading EEM application volatile data. EEM application volatile data can be published by a Cisco IOS XR software process that uses the EEM application publish API. EEM application volatile data cannot be published by an EEM policy.

Note Currently there are no Cisco IOS XR software processes that publish application volatile data.

Syntax

```
appl_read name ? length ?
```

Arguments

name	(Mandatory) Name of the application published string data.
length	(Mandatory) Length of the string data to read. Must be an integer number between 1 and 4294967295, inclusive.

Result String

```
data %s
```

Where data is the application published string data to be read.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

This error means that the application event detector info key or other ID was not found.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

appl_reqinfo

Retrieves previously saved information from the Embedded Event Manager (EEM). This Tcl command extension provides support for retrieving information from EEM that has been previously saved with a unique key, which must be specified in order to retrieve the information. Note that retrieving the information deletes it from EEM. It must be resaved if it is to be retrieved again.

Syntax

```
appl_reqinfo key ?
```

Arguments

key	(Mandatory) String key of the data.

Result String

```
data %s
```

Where data is the application string data to be retrieved.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

This error means that the application event detector info key or other ID was not found.

appl_setinfo

Saves information in the EEM. This Tcl command extension provides support for saving information in the EEM that can be retrieved later by the same policy or by another policy. A unique key must be specified. This key allows the information to be retrieved later.

Syntax

```
appl_setinfo key ? data ?
```

Arguments

key	(Mandatory) String key of the data.
data	(Mandatory) Application string data to save.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  8)    FH_EDUPLICATEKEY  (duplicate appl info key)
```

This error means that the application event detector info key or other ID was a duplicate.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 34)    FH_EMAXLEN  (maximum length exceeded)
```

This error means that the object length or number exceeded the maximum.

```
(_cerr_sub_err = 43)    FH_EBADLENGTH  (bad API length)
```

This error means that the API message length was invalid.

counter_modify

Modifies a counter value.

Syntax

```
counter_modify event_id ? val ? op nop|set|inc|dec
```

Arguments

event_id	(Mandatory) Counter event ID returned by the **register_counter** Tcl command extension. Must be an integer between 0 and 4294967295, inclusive.
val	(Mandatory) • If op is set, this argument represents the counter value that is to be set. • If op is inc, this argument is the value by which to increment the counter. • If op is dec, this argument is the value by which to decrement the counter.

op	(Mandatory) • nop—Retrieves the current counter value. • set—Sets the counter value to the given value. • inc—Increments the counter value by the given value. • dec—Decrements the counter value by the given value.

Result String

```
val_remain %d
```

Where val_remain is the current value of the counter.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 30)    FH_ECTBADOPER  (bad counter threshold operator)
```

This error means that the counter event detector set or modify operator was invalid.

fts_get_stamp

Returns the time period elapsed since the last software boot. Use this Tcl command extension to return the number of nanoseconds since boot in an array “nsec nnnn” where nnnn is the number of nanoseconds.

Syntax

```
fts_get_stamp
```

Arguments

None

Result String

```
nsec %d
```

Where nsec is the number of nanoseconds since boot.

Set _cerrno

No

register_counter

Registers a counter and returns a counter event ID. This Tcl command extension is used by a counter publisher to perform this registration before using the event ID to manipulate the counter.

Syntax

```
register_counter name ?
```

Arguments

name	(Mandatory) The name of the counter to be manipulated.

Result String

```
event_id %d
event_spec_id %d
```

Where event_id is the counter event ID for the specified counter; it can be used to manipulate the counter by the **unregister_counter** or **counter_modify** Tcl command extensions. The event_spec_id argument is the event specification ID for the specified counter.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  4)    FH_EINITONCE  (Init() is not yet done, or done twice.)
```

This error means that the request to register the specific event was made before the EEM event detector had completed its initialization.

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

This error means that the event type specified in the internal event specification was invalid.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 10)    FH_ECORRUPT  (internal EEM API context is corrupt)
```

This error means that the internal EEM API context structure is corrupt.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

This error means that the event ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 16)    FH_EBADFMPPTR  (bad ptr to fh_p data structure)
```

This error means that the context pointer that is used with each EEM API call is incorrect.

```
(_cerr_sub_err = 17)    FH_EBADADDRESS  (bad API control block address)
```

This error means that a control block address that was passed in the EEM API was incorrect.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 25)    FH_ESUBSEXCEED  (number of subscribers exceeded)
```

This error means that the number of timer or counter subscribers exceeded the maximum.

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

This error means that the subscriber index was invalid.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

This error means that the EEM event detector that handles this request is not available.

register_timer

Registers a timer and returns a timer event ID. This Tcl command extension is used by a timer publisher to perform this registration before using the event ID to manipulate the timer if it does not use the **event_register_timer** command extension to register as a publisher and subscriber.

Syntax

```
register_timer watchdog|countdown|absolute|cron name ?
```

Arguments

name	(Mandatory) Name of the timer to be manipulated.

Result String

```
event_id %u
```

Where event_id is the timer event ID for the specified timer (can be used to manipulate the timer by the **timer_arm** or **timer_cancel** command extensions).

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  4)    FH_EINITONCE  (Init() is not yet done, or done twice.)
```

This error means that the request to register the specific event was made before the EEM event detector had completed its initialization.

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

This error means that the event type specified in the internal event specification was invalid.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 10)    FH_ECORRUPT  (internal EEM API context is corrupt)
```

This error means that the internal EEM API context structure is corrupt.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 16)    FH_EBADFMPPTR  (bad ptr to fh_p data structure)
```

This error means that the context pointer that is used with each EEM API call is incorrect.

```
(_cerr_sub_err = 17)    FH_EBADADDRESS  (bad API control block address)
```

This error means that a control block address that was passed in the EEM API was incorrect.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 25)    FH_ESUBSEXCEED  (number of subscribers exceeded)
```

This error means that the number of timer or counter subscribers exceeded the maximum.

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

This error means that the subscriber index was invalid.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

This error means that the EEM event detector that handles this request is not available.

timer_arm

Arms a timer. The type could be CRON, watchdog, countdown, or absolute.

Syntax

```
timer_arm event_id ? cron_entry ?|time ?
```

Arguments

event_id	(Mandatory)Timer event ID returned by the **register_timer** command extension. Must be an integer between 0 and 4294967295, inclusive.
cron_entry	(Mandatory) Must exist if the timer type is CRON. Must not exist for other types of timer. CRON timer specification uses the format of the CRON table entry.

time	(Mandatory) Must exist if the timer type is not CRON. Must not exist if the timer type is CRON. For watchdog and countdown timers, the number of seconds and milliseconds until the timer expires; for an absolute timer, the calendar time of the expiration time (specified in SSSSSSSSS[.MMM] format, where SSSSSSSSS must be an integer representing seconds between 0 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). An absolute expiration date is the number of seconds and milliseconds since January 1, 1970. If the date specified has already passed, the timer expires immediately.

Result String

```
sec_remain %ld msec_remain %ld
```

Where sec_remain and msec_remain are the remaining time before the next expiration of the timer.

Note A value of 0 is returned for the sec_remain and msec_remain arguments if the timer type is CRON.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

This error means that the event type specified in the internal event specification was invalid.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

This error means that the event ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 27)    FH_ETMDELAYZR  (zero delay time)
```

This error means that the time specified to arm a timer was zero.

```
(_cerr_sub_err = 42)    FH_ENOTREGISTERED  (request for event spec that is unregistered)
```

This error means that the event was not registered.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

This error means that the EEM event detector that handles this request is not available.

timer_cancel

Cancels a timer.

Syntax

```
timer_cancel event_id ?
```

Arguments

event_id	(Mandatory) Timer event ID returned by the **register_timer** command extension. Must be an integer between 0 and 4294967295, inclusive.

Result String

```
sec_remain %ld msec_remain %ld
```

Where sec_remain and msec_remain are the remaining time before the next expiration of the timer.

Note A value of 0 will be returned for sec_remain and msec_remain if the timer type is CRON.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

This error means that the event type specified in the internal event specification was invalid.

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

This error means that the application event detector info key or other ID was not found.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

This error means that the event ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

This error means that the EEM event detector that handles this request is not available.

unregister_counter

Unregisters a counter. This Tcl command extension is used by a counter publisher to unregister a counter that was previously registered with the **register_counter** Tcl command extension.

Syntax

```
unregister_counter event_id ? event_spec_id ?
```

Arguments

event_id	(Mandatory) Counter event ID returned by the **register_counter** command extension. Must be an integer between 0 and 4294967295, inclusive.
event_spec_id	(Mandatory) Counter event specification ID for the specified counter returned by the **register_counter** command extension. Must be an integer between 0 and 4294967295, inclusive.

Result String

None

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

This error means that the event specification ID could not be matched when the event was being registered or that an event detector internal event structure is corrupt.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

This error means that the subscriber index was invalid.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

This error means that the EEM event detector that handles this request is not available.

Embedded Event Manager System Information Tcl Command Extensions

Note All EEM system information commands—**sys_reqinfo _xxx**—have the Set _cerrno section set to **yes**.

sys_reqinfo_cpu_all

Queries the CPU utilization of the top processes (both POSIX processes and IOS processes) during a specified time period and in a specified order. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
sys_reqinfo_cpu_all order cpu_used [sec ?] [msec ?] [num ?]
```

Arguments

order	(Mandatory) Order used for sorting the CPU utilization of processes.
cpu_used	(Mandatory) Specifies that the average CPU utilization, for the specified time window, will be sorted in descending order.
secmsec	(Optional) Time period, in seconds and milliseconds, during which the average CPU utilization is calculated. Must be integers in the range from 0 to 4294967295. If not specified, or if both sec and msec are specified as 0, the most recent CPU sample is used.
num	(Optional) Number of entries from the top of the sorted list of processes to be displayed. Must be an integer in the range from 1 to 4294967295. Default value is 5.

Result String

```
rec_list {{process CPU info string 0},{process CPU info string 1}, ...}
```

Where each process CPU info string is:

```
pid %u name {%s} cpu_used %u
```

rec_list	Marks the start of the process CPU information list.
pid	Process ID.
name	Process name.
cpu_used	Specifies that if sec and msec are specified with a number greater than zero, the average percentage is calculated from the process CPU utilization during the specified time period. If sec and msec are both zero or not specified, the average percentage is calculated from the process CPU utilization in the latest sample.

Set _cerrno

Yes

sys_reqinfo_crash_history

Queries the crash information of all processes that have ever crashed. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
sys_reqinfo_crash_history
```

Arguments

None

Result String

```
rec_list {{crash info string 0},{crash info string 1}, ...}

Where each crash info string is:

job_id %u name {%s} respawn_count %u fail_count %u dump_count %u
inst_id %d exit_status 0x%x exit_type %d proc_state {%s} component_id 0x%x
crash_time_sec %ld crash_time_msec %ld
```

job_id	System manager assigned job ID for the process. An integer between 1 and 4294967295, inclusive.
name	Process name.
respawn_count	Total number of restarts for the process.
fail_count	Number of restart attempts of the process. This count is reset to zero when the process is successfully restarted.
dump_count	Number of core dumps performed.
inst_id	Process instance ID.
exit_status	Last exit status of the process.
exit_type	Last exit type.
proc_state	Sysmgr process states. One of the following: error, forced_stop, hold, init, ready_to_run, run, run_rnode, stop, waitEOltimer, wait_rnode, wait_spawntimer, wait_tpl.
component_id	Version manager assigned component ID for the component to which the process belongs.
crash_time_sec crash_time_msec	Seconds and milliseconds since January 1, 1970, which represent the last time the process crashed.

Set _cerrno

Yes

sys_reqinfo_mem_all

Queries the memory usage of the top processes (both POSIX and IOS) during a specified time period and in a specified order. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
sys_reqinfo_mem_all order allocates|increase|used [sec ?] [msec ?] [num ?]
```

Arguments

order	(Mandatory) Order used for sorting the memory usage of processes.
allocates	(Mandatory) Specifies that the memory usage is sorted by the number of process allocations during the specified time window, and in descending order.
increase	(Mandatory) Specifies that the memory usage is sorted by the percentage of process memory increase during the specified time window, and in descending order.
used	(Mandatory) Specifies that the memory usage is sorted by the current memory used by the process.
secmsec	(Optional) Time period, in seconds and milliseconds, during which the process memory usage is calculated. Must be integers in the range from 0 to 4294967295. If both sec and msec are specified and are nonzero, the number of allocations is the difference between the number of allocations in the oldest and latest samples collected in the time period. The percentage is calculated as the the percentage difference between the memory used in the oldest and latest samples collected in the time period. If not specified, or if both sec and msec are specified as 0, the first sample ever collected is used as the oldest sample; that is, the time period is set to be the time from startup until the current moment.
num	(Optional) Number of entries from the top of the sorted list of processes to be displayed. Must be an integer in the range from 1 to 4294967295. Default value is 5.

Result String

```
rec_list {{process mem info string 0},{process mem info string 1}, ...}
```

Where each process mem info string is:

```
pid %u name {%s} delta_allocs %d initial_alloc %u current_alloc %u percent_increase %d
```

rec_list	Marks the start of the process memory usage information list.
pid	Process ID.
name	Process name.
delta_allocs	Specifies the difference between the number of allocations in the oldest and latest samples collected in the time period.

initial_alloc	Specifies the amount of memory, in kilobytes, used by the process at the start of the time period.
current_alloc	Specifies the amount of memory, in kilobytes, currently used by the process.
percent_increase	Specifies the percentage difference between the memory used in the oldest and latest samples collected in the time period. The percentage difference can be expressed as current_alloc minus initial_alloc times 100 and divided by initial_alloc.

Set _cerrno

Yes

sys_reqinfo_proc

Queries the information about a single POSIX process. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
sys_reqinfo_proc job_id ?
```

Arguments

job_id	(Mandatory) System manager assigned job ID for the process. Must be an integer between 1 and 4294967295, inclusive.

Result String

```
job_id %u component_id 0x%x name {%s} helper_name {%s} helper_path {%s} path {%s}
node_name {%s} is_respawn %u is_mandatory %u is_hold %u dump_option %d
max_dump_count %u respawn_count %u fail_count %u dump_count %u
last_respawn_sec %ld last_respawn_msec %ld inst_id %u proc_state %s
level %d exit_status 0x%x exit_type %d
```

job_id	System manager assigned job ID for the process. An integer between 1 and 4294967295, inclusive.
component_id	Version manager assigned component ID for the component to which the process belongs.
name	Process name.
helper_name	Helper process name.
helper_path	Executable path of the helper process.
path	Executable path of the process.
node_name	System manager assigned node name for the node to which the process belongs.

is_respawn	Flag that specifies that the process can be respawned.
is_mandatory	Flag that specifies that the process must be alive.
is_hold	Flag that specifies that the process is spawned until called by the API.
dump_option	Core dumping options.
max_dump_count	Maximum number of core dumping permitted.
respawn_count	Total number of restarts for the process.
fail_count	Number of restart attempts of the process. This count is reset to zero when the process is successfully restarted.
dump_count	Number of core dumps performed.
last_respawn_seclast_respawn_msec	Seconds and milliseconds in POSIX timer units since January 1, 1970, which represent the last time the process was started.
inst_id	Process instance ID.
proc_state	Sysmgr process states. One of the following: error, forced_stop, hold, init, ready_to_run, run, run_rnode, stop, waitEOltimer, wait_rnode, wait_spawntimer, wait_tpl.
level	Process run level.
exit_status	Last exit status of the process.
exit_type	Last exit type.

Set _cerrno

Yes

sys_reqinfo_proc_all

Queries the information of all POSIX processes. This Tcl command extension is supported only in Software Modularity images.

Syntax

```
sys_reqinfo_proc_all
```

Arguments

None

Result String

```
rec_list {{process info string 0}, {process info string 1},...}
```

Where each process info string is the same as the result string of the **sysreq_info_proc** Tcl command extension.

Set _cerrno

Yes

sys_reqinfo_proc_version

Queries the version of the given process.

Syntax

```
sys_reqinfo_proc_version [job_id ?]
```

Arguments

job_id	(Mandatory) System manager assigned job ID for the process. The integer number must be inclusively between 1 and 2147483647.

Result String

```
version_id %02d.%02d.%04d
```

Where version_id is the version manager that is assigned the version number of the process.

Set _cerrno

Yes

sys_reqinfo_routername

Queries the router name.

Syntax

```
sys_reqinfo_routername
```

Arguments

None

Result String

```
routername %s
```

Where routername is the name of the router.

Set _cerrno

Yes

sys_reqinfo_syslog_freq

Queries the frequency information of all syslog events.

Syntax

```
sys_reqinfo_syslog_freq
```

Arguments

None

Result String

```
rec_list {{event frequency string 0}, {log freq str 1}, ...}
```

Where each event frequency string is:

```
time_sec %ld time_msec %ld match_count %u raise_count %u occurs %u
period_sec %ld period_msec %ld pattern {%s}
```

time_sectime_msec	Seconds and milliseconds in POSIX timer units since January 1, 1970, which represent the time the last event was raised.
match_count	Number of times that a syslog message matches the pattern specified by this syslog event specification since event registration.
raise_count	Number of times that this syslog event was raised.
occurs	Number of occurrences needed in order to raise the event; if not specified, the event is raised on the first occurrence.
period_secperiod_msec	Number of occurrences must occur within this number of POSIX timer units in order to raise the event; if not specified, the period check does not apply.
pattern	Regular expression used to perform syslog message pattern matching.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

This error means that an internal EEM request for memory failed.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 45)    FH_ESEQNUM  (sequence or workset number out of sync)
```

This error means that the event detector sequence or workset number was invalid.

```
(_cerr_sub_err = 46)    FH_EREGEMPTY  (registration list is empty)
```

This error means that the event detector registration list was empty.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

sys_reqinfo_syslog_history

Queries the history of the specified syslog message.

Syntax

```
sys_reqinfo_syslog_history
```

Arguments

None

Result String

```
rec_list {{log hist string 0}, {log hist str 1}, ...}
```

Where each log hist string is:

```
time_sec %ld time_msec %ld msg {%s}
```

time_sec time_msec	Seconds and milliseconds since January 1, 1970, which represent the time the message was logged.
msg	Syslog message.

Set _cerrno

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

This error means that an internal EEM event detector pointer was null when it should have contained a value.

```
(_cerr_sub_err = 44)    FH_EHISTEMPTY  (history list is empty)
```

This error means that the history list was empty.

```
(_cerr_sub_err = 45)    FH_ESEQNUM  (sequence or workset number out of sync)
```

This error means that the event detector sequence or workset number was invalid.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

This error means that the event detector was unavailable.

sys_reqinfo_stat

Queries the value of the statistic entity that is specified by name, and optionally the first modifier and the second modifier.

Syntax

```
sys_reqinfo_stat [name ?][mod1 ?][mod2 ?]
```

Arguments

name	(Mandatory) Statistics data element name.
mod_1	(Optional) Statistics data element modifier 1.
mod_2	(Optional) Statistics data element modifier 2.

Result String

```
name %s value %s
```

name	Statistics data element name.
value	Value string of the statistics data element.

Set _cerrno

Yes

sys_reqinfo_snmp

Queries the value of the entity specified by a Simple Network Management Protocol (SNMP) object ID.

Syntax

```
sys_reqinfo_snmp oid ? get_type exact|next
```

Arguments

oid	(Mandatory) SNMP OID in dot notation (for example, 1.3.6.1.2.1.2.1.0).
get_type	(Mandatory) Type of SNMP get operation that needs to be applied to the specified oid. If the get_type is "exact," the value of the specified oid is retrieved; if the get_type is "next," the value of the lexicographical successor to the specified oid is retrieved.

Result String

```
oid {%s} value {%s}
```

oid	SNMP OID.
value	Value string of the associated SNMP data element.

sys_reqinfo_snmp_trap

This command is used to send a trap.

Syntax

```
sys_reqinfo_snmp_trap enterprise_oid ent-oid generic_trapnum gen-trapnum specific_trapnum
spe-trapnum
 trap_oid oid trap_var varname
```

- Use the ***enterprise_oid*** argument to specify the enterprise oid of the trap.
- Use the ***generic_trapnum*** argument to specify generic trap number of the trap.
- Use the ***specific_trapnum*** argument to specify specific trap number of the trap.
- Use the ***trap_oid*** argument to specify oid of the trap to send.
- Use the ***trap_var*** argument to specify the variable of oid(s) to send.

Example

sys_reqinfo_snmp_trap **enterprise_oid** 1.3.6.1.4.1.9.9.41.2 **generic_trapnum** 6 **specific_trapnum** 1 **trap_oid** 1.3.6.1.4.1.9.9.41.2.0.1 **trap_var** var1

sys_reqinfo_snmp_trapvar

This command is used to setup an array of oid and value given a trap variable. Similar to IOS, the trap variable can contain a list of 10 multiple oids and values.

Syntax

```
sys_reqinfo_snmp_trapvar var varname oid oid int|uint|counter|gauge|octet|string|ipv4  value
```

- Use the ***var*** argument to specify the trap variable name.

- Use the ***oid*** argument to specify the oid of the trap.

Example

sys_reqinfo_snmp_trapvar **var** var1 **oid** 1.3.6.1.4.1.9.9.41.1.2.3.1.3 **int** 4

SMTP Library Command Extensions

All Simple Mail Transfer Protocol (SMTP) library command extensions belong to the ::cisco::lib namespace.

To use this library, the user needs to provide an e-mail template file. The template file can include Tcl global variables so that the e-mail service and the e-mail text can be configured through the **event manager environment** Cisco IOS XR software command-line interface (CLI) configuration command. There are commands in this library to substitute the global variables in the e-mail template file and to send the desired e-mail context with the To address, CC address, From address, and Subject line properly configured using the configured e-mail server.

E-Mail Template

The e-mail template file has the following format:

```
Mailservername:<space><the list of candidate SMTP server addresses>
From:<space><the e-mail address of sender>
To:<space><the list of e-mail addresses of recipients>
Cc:<space><the list of e-mail addresses that the e-mail will be copied to>
Subject:<subject line>
<a blank line>
<body>
```

Note The template normally includes Tcl global variables to be configured.

The following is a sample e-mail template file:

```
Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc: $_email_cc
Subject: From router $routername: Process terminated

process name: $process_name
subsystem: $sub_system
exit status: $exit_status
respawn count: $respawn_count
```

Exported Tcl Command Extensions

smtp_send_email

Given the text of an e-mail template file with all global variables already substituted, sends the e-mail out using Simple Mail Transfer Protocol (SMTP). The e-mail template specifies the candidate mail server addresses, To addresses, CC addresses, From address, subject line, and e-mail body.

Note A list of candidate e-mail servers can be provided so that the library will try to connect the servers on the list one by one until it can successfully connect to one of them.

Syntax

```
smtp_send_email text
```

Arguments

text	(Mandatory) Text of an e-mail template file with all global variables already substituted.

Result String

None

Set _cerrno

- Wrong 1st line format—Mailservername:list of server names.
- Wrong 2nd line format—From:from-address.
- Wrong 3rd line format—To:list of to-addresses.
- Wrong 4th line format—CC:list of cc-addresses.
- Error connecting to mail server:—$sock closed by remote server (where $sock is the name of the socket opened to the mail server).
- Error connecting to mail server:—$sock reply code is $k instead of the service ready greeting (where $sock is the name of the socket opened to the mail server; $k is the reply code of $sock).
- Error connecting to mail server:—cannot connect to all the candidate mail servers.
- Error disconnecting from mail server:—$sock closed by remote server (where $sock is the name of the socket opened to the mail server).

Sample Scripts

After all needed global variables in the e-mail template are defined:

```
if [catch {smtp_subst [file join $tcl_library email_template_sm]} result] {
    puts stderr $result
    exit 1
}
if [catch {smtp_send_email $result} result] {
    puts stderr $result
    exit 1
}
```

smtp_subst

Given an e-mail template file e-mail_template, substitutes each global variable in the file by its user-defined value. Returns the text of the file after substitution.

Syntax

```
smtp_subst e-mail_template
```

Arguments

e-mail_template	(Mandatory) Name of an e-mail template file in which global variables need to be substituted by a user-defined value. An example filename could be /disk0://example.template which represents a file named example.template in a top-level directory on an ATA flash disk in slot 0.

Result String

The text of the e-mail template file with all the global variables substituted.

Set _cerrno

- cannot open e-mail template file
- cannot close e-mail template file

CLI Library Command Extensions

All command-line interface (CLI) library command extensions belong to the ::cisco::eem namespace.

This library provides users the ability to run CLI commands and get the output of the commands in Tcl. Users can use commands in this library to spawn an exec and open a virtual terminal channel to it, write the command to execute to the channel so that the command will be executed by exec, and read back the output of the command.

There are two types of CLI commands: interactive commands and non-interactive commands.

For interactive commands, after the command is entered, there will be a “Q&A” phase in which the router will ask for different user options, and the user is supposed to enter the answer for each question. Only after all the questions have been answered properly will the command run according to the user’s options until completion.

For noninteractive commands, once the command is entered, the command will run to completion. To run different types of commands using an EEM script, different CLI library command sequences should be used, which are documented in the Using the CLI Library to Run a Noninteractive Command, on page 132 and in the Using the CLI Library to Run an Interactive Command, on page 133.

Exported Tcl Command Extensions

cli_close

Closes the exec process and releases the VTY and the specified channel handler connected to the command-line interface (CLI).

Syntax

```
cli_close fd tty_id
```

Arguments

fd	(Mandatory) The CLI channel handler.
tty_id	(Mandatory) The TTY ID returned from the **cli_open** command extension.

Result String

None

Set _cerrno

Cannot close the channel.

cli_exec

Writes the command to the specified channel handler to execute the command. Then reads the output of the command from the channel and returns the output.

Syntax

```
cli_exec fd cmd
```

Arguments

fd	(Mandatory) The command-line interface (CLI) channel handler.
cmd	(Mandatory) The CLI command to execute.

Result String

The output of the CLI command executed.

Set _cerrno

Error reading the channel.

cli_get_ttyname

Returns the real and pseudo tty names for a given TTY ID.

Syntax

```
cli_get_ttyname tty_id
```

Arguments

tty_id	(Mandatory) The TTY ID returned from the **cli_open** command extension.

Result String

```
pty %s tty %s
```

Set _cerrno

None

cli_open

Calls **vty_connect** to open a vty channel to an EXEC session and passes the username and associated taskmap value for the script to be executed. The EXEC session first attempts to authorize the user using whichever AAA method was configured for the EEM vty pool. If this fails, the EXEC session attempts to authorize the user using the taskmap value. Returns an array including the vty channel handler.

Note To configure authorization methods for the EEM vty pool, you must configure an AAA method list, create a vty line template for the EEM vty pool, enable the AAA method list on the vty line template, and configure the EEM vty pool to use the vty line template you created. For information about AAA authorization configuration, see *Configuring AAA Services* module of *System Security Configuration Guide for Cisco CRS Routers*.

Note Each call to **cli_open** initiates a Cisco IOS XR software EXEC session that allocates a Cisco IOS XR software vty. The vty remains in use until the cli_close routine is called. Vtys are allocated from the pool of vtys that are configured using the **line vty vty-pool** CLI configuration command. Be aware that the cli_open routine fails when two or fewer vtys are available, preserving the remaining vtys for Telnet use.

Syntax

```
cli_open
```

Arguments

None

Result String

```
"tty_id {%s} pty {%d} tty {%d} fd {%d}"
```

Event Type	Description
tty_id	TTY ID.
pty	PTY device name.
tty	TTY device name.
fd	CLI channel handler.

Set _cerrno

- Cannot get pty for EXEC.
- Cannot create an EXEC CLI session.
- Error reading the first prompt.

cli_read

Reads the command output from the specified command-line interface (CLI) channel handler until the pattern of the router prompt occurs in the contents read. Returns all the contents read up to the match.

Syntax

```
cli_read fd
```

Arguments

fd	(Mandatory) CLI channel handler.

Result String

All the contents read.

Set _cerrno

Cannot get router name.

Note This Tcl command extension blocks waiting for the router prompt to show up in the contents read.

cli_read_drain

Reads and drains the command output of the specified command-line interface (CLI) channel handler. Returns all the contents read.

Syntax

```
cli_read_drain fd
```

Arguments

fd	(Mandatory) The CLI channel handler.

Result String

All the contents read.

Set _cerrno

None

cli_read_line

Reads one line of the command output from the specified command-line interface (CLI) channel handler. Returns the line read.

Syntax

```
cli_read_line fd
```

Arguments

fd	(Mandatory) CLI channel handler.

Result String

The line read.

Set _cerrno

None

Note This Tcl command extension blocks waiting for the end of line to show up in the contents read.

cli_read_pattern

Reads the command output from the specified command-line interface (CLI) channel handler until the pattern that is to be matched occurs in the contents read. Returns all the contents read up to the match.

Note The pattern matching logic attempts a match by looking at the command output data as it is delivered from the Cisco IOS XR software command. The match is always done on the most recent 256 characters in the output buffer unless there are fewer characters available, in which case the match is done on fewer characters. If more than 256 characters in the output buffer are required for the match to succeed, the pattern will not match.

Syntax

```
cli_read_pattern fd ptn
```

Arguments

fd	(Mandatory) CLI channel handler.
ptn	(Mandatory) Pattern to be matched when reading the command output from the channel.

Result String

All the contents read.

Set _cerrno

None

Note This Tcl command extension blocks waiting for the specified pattern to show up in the contents read.

cli_write

Writes the command that is to be executed to the specified CLI channel handler. The CLI channel handler executes the command.

Syntax

```
cli_write fd cmd
```

Arguments

fd	(Mandatory) The CLI channel handler.
cmd	(Mandatory) The CLI command to execute.

Result String

None

Set _cerrno

None

Sample Usage

As an example, use configuration CLI commands to bring up Ethernet interface 1/0:

```
if [catch {cli_open} result] {
puts stderr $result
exit 1
} else {
array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "config t"} result] {
puts stderr $result
exit 1
}
if [catch {cli_exec $cli1(fd) "interface Ethernet1/0"} result] {
puts stderr $result
exit 1
}
if [catch {cli_exec $cli1(fd) "no shut"} result] {
puts stderr $result
exit 1
}
if [catch {cli_exec $cli1(fd) "end"} result] {
puts stderr $result
exit 1
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} } result] {
puts stderr $result
exit 1
```

Using the CLI Library to Run a Noninteractive Command

To run a noninteractive command, use the **cli_exec** command extension to issue the command, and then wait for the complete output and the router prompt. For example, the following shows the use of configuration CLI commands to bring up Ethernet interface 1/0:

```
if [catch {cli_open} result] {
error $result $errorInfo
} else {
set fd $result
}
if [catch {cli_exec $fd "config t"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "interface Ethernet1/0"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "no shut"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "end"} result] {
error $result $errorInfo
}
if [catch {cli_close $fd} result] {
error $result $errorInfo
}
```

Using the CLI Library to Run an Interactive Command

To run interactive commands, three phases are needed:

- Phase 1: Issue the command using the **cli_write** command extension.
- Phase 2: Q&A Phase. Use the **cli_read_pattern** command extension to read the question (the regular pattern that is specified to match the question text) and the **cli_write** command extension to write back the answers alternately.
- Phase 3: Noninteractive phase. All questions have been answered, and the command will run to completion. Use the **cli_read** command extension to wait for the complete output of the command and the router prompt.

For example, use CLI commands to do squeeze bootflash: and save the output of this command in the Tcl variable cmd_output.

```
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}

# Phase 1: issue the command
if [catch {cli_write $cli1(fd) "squeeze bootflash:"} result] {
error $result $errorInfo
}

# Phase 2: Q&A phase
# wait for prompted question:
# All deleted files will be removed. Continue? [confirm]
if [catch {cli_read_pattern $cli1(fd) "All deleted"} result] {
error $result $errorInfo
}
# write a newline character
if [catch {cli_write $cli1(fd) "\n"} result] {
error $result $errorInfo
}
# wait for prompted question:
# Squeeze operation may take a while. Continue? [confirm]
if [catch {cli_read_pattern $cli1(fd) "Squeeze operation"} result] {
error $result $errorInfo
}
# write a newline character
if [catch {cli_write $cli1(fd) "\n"} result] {
error $result $errorInfo
}

# Phase 3: noninteractive phase
# wait for command to complete and the router prompt
if [catch {cli_read $cli1(fd) } result] {
error $result $errorInfo
} else {
set cmd_output $result
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
error $result $errorInfo
}
```

The following example causes a router to be reloaded using the CLI **reload** command. Note that the EEM **action_reload** command accomplishes the same result in a more efficient manner, but this example is presented to illustrate the flexibility of the CLI library for interactive command execution.

```
# 1. execute the reload command
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}
if [catch {cli_write $cli1(fd) "reload"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_read_pattern $cli1(fd) ".*(System configuration has been modified. Save\\\?
 \\\[yes/no\\\]: )"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_write $cli1(fd) "no"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_read_pattern $cli1(fd) ".*(Proceed with reload\\\? \\\[confirm\\\])"} result]
 {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_write $cli1(fd) "y"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}
```

Tcl Context Library Command Extensions

All the Tcl context library command extensions belong to the ::cisco::eem namespace.

Exported Commands

context_retrieve

Retrieves Tcl variable(s) identified by the given context name, and possibly the scalar variable name, the array variable name, and the array index. Retrieved information is automatically deleted.

Note Once saved information is retrieved, it is automatically deleted. If that information is needed by another policy, the policy that retrieves it (using the **context_retrieve** command extension) should also save it again (using the **context_save** command extension).

Syntax

```
context_retrieve ctxt [var] [index_if_array]
```

Arguments

ctxt	(Mandatory) Context name.
var	(Optional) Scalar variable name or array variable name. Defaults to a null string if this argument is not specified.
index_if_array	(Optional) Array index.

Note The *index_if_array* argument is ignored when the *var* argument is a scalar variable.

If *var* is unspecified, retrieves the whole variable table saved in the context.

If *var* is specified and *index_if_array* is not specified, or if *index_if_array* is specified but *var* is a scalar variable, retrieves the value of *var*.

If *var* is specified, and *index_if_array* is specified, and *var* is an array variable, retrieves the value of the specified array element.

Result String

Resets the Tcl global variables to the state that they were in when the save was performed.

Set _cerrno

- A string displaying _cerrno, _cerr_sub_num, _cerr_sub_err, _cerr_posix_err, _cerr_str due to appl_reqinfo error.
- Variable is not in the context.

Sample Usage

The following examples show how to use the **context_save** and **context_retrieve** command extension functionality to save and retrieve data. The examples are shown in save and retrieve pairs.

Example 1: Save

If var is unspecified or if a pattern if specified, saves multiple variables to the context.

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

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

set testvara 123
set testvarb 345
set testvarc 789
if {[catch {context_save TESTCTX "testvar*"} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
```

```
} else {
      action_syslog msg "context_save succeeded"
}
```

Example 1: Retrieve

If var is unspecified, retrieves multiple variables from the context.

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

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

if {[catch {foreach {var value} [context_retrieve TESTCTX] {set $var $value}} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvara]} {
      action_syslog msg "testvara exists and is $testvara"
} else {
      action_syslog msg "testvara does not exist"
}
if {[info exists testvarb]} {
      action_syslog msg "testvarb exists and is $testvarb"
} else {
      action_syslog msg "testvarb does not exist"
}
if {[info exists testvarc]} {
      action_syslog msg "testvarc exists and is $testvarc"
} else {
      action_syslog msg "testvarc does not exist"
}
```

Example 2: Save

If var is specified, saves the value of var.

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

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

set testvar 123
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

Example 2: Retrieve

If var is specified and index_if_array is not specified, or if index_if_array is specified but var is a scalar variable, retrieves the value of var.

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

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

if {[catch {set testvar [context_retrieve TESTCTX testvar]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is $testvar"
} else {
      action_syslog msg "testvar does not exist"
}
```

Example 3: Save

If var is specified, saves the value of var even if it is an array.

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

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

array set testvar "testvar1 ok testvar2 not_ok"
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

Example 3: Retrieve

If var is specified, and index_if_array is not specified, and var is an array variable, retrieves the entire array.

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

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

if {[catch {array set testvar [context_retrieve TESTCTX testvar]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is [array get testvar]"
} else {
      action_syslog msg "testvar does not exist"
}
```

Example 4: Save

If var is specified, saves the value of var even if it is an array.

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

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

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

array set testvar "testvar1 ok testvar2 not_ok"
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

Example 4: Retrieve

If var is specified, and index_if_array is specified, and var is an array variable, retrieves the specified array element value.

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

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

if {[catch {set testvar [context_retrieve TESTCTX testvar testvar1]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is $testvar"
} else {
      action_syslog msg "testvar doesn't exist"
}
```

context_save

Saves Tcl variables that match a given pattern in current and global namespaces with the given context name as identification. Use this Tcl command extension to save information outside of a policy. Saved information can be retrieved by a different policy using the **context_retrieve** command extension.

Note Once saved information is retrieved, it is automatically deleted. If that information is needed by another policy, the policy that retrieves it (using the **context_retrieve** command extension) should also save it again (using the **context_save** command extension).

Syntax

```
context_save ctxt [pattern]
```

Arguments

ctxt	(Mandatory) Context name.

pattern	(Optional) Glob-style pattern as used by the **string match** Tcl command. If this argument is not specified, the pattern defaults to the wildcard *. There are three constructs used in glob patterns: • * = all characters • ? = 1 character • [abc] = match one of a set of characters

Result String

None

Set _cerrno

A string displaying _cerrno, _cerr_sub_num, _cerr_sub_err, _cerr_posix_err,_cerr_str due to appl_setinfo error.

Sample Usage

For examples showing how to use the **context_save** and **context_retrieve** command extension functionality to save and retrieve data, see the Sample Usage, on page 135.