Embedded Event Manager Configuration Guide, Cisco IOS XE Release 3SE (Catalyst 3850 Switches)

First Published: November 20, 2012

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: http://www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2015 Cisco Systems, Inc. All rights reserved.

CONTENTS

CHAPTER 1 **Embedded Event Manager Overview** 1

Finding Feature Information 1

Information About Embedded Event Manager 1

Embedded Event Manager 1

Embedded Event Manager 1.0 3

Embedded Event Manager 2.0 3

Embedded Event Manager 2.1 4

Embedded Event Manager 2.1 (Software Modularity) 4

Embedded Event Manager 2.2 5

Embedded Event Manager 2.3 5

Embedded Event Manager 2.4 6

Embedded Event Manager 3.0 6

Embedded Event Manager 3.1 7

Embedded Event Manager 3.2 8

EEM Event Detectors Available by Cisco IOS Release 8

Event Detectors 10

EEM Actions Available by Cisco IOS Release 14

Embedded Event Manager Actions 16

Embedded Event Manager Environment Variables 17

Embedded Event Manager Policy Creation 19

Where to Go Next 20

Additional References 20

Feature Information for Embedded Event Manager 3.2 Overview 21

CHAPTER 2 **Writing Embedded Event Manager Policies Using the Cisco IOS CLI** 23

Finding Feature Information 23

Prerequisites for Writing EEM Policies Using the Cisco IOS CLI 24

Information About Writing EEM Policies Using the Cisco IOS CLI 24

Embedded Event Manager Policies 24
EEM Applet 24
EEM Script 25
Embedded Event Manager Built-In Environment Variables Used in EEM Applets 25
How to Write EEM Policies Using the Cisco IOS CLI 37
Registering and Defining an Embedded Event Manager Applet 37
EEM Environment Variables 37
Alphabetical Order of EEM Action Labels 38
Troubleshooting Tips 41
Registering and Defining an Embedded Event Manager Policy to Run Manually 42
Unregistering Embedded Event Manager Policies 43
Suspending All Embedded Event Manager Policy Execution 45
Configuring and Tracking a Stub Object Using Embedded Event Manager 46
Enhanced Object Tracking 46
Displaying Embedded Event Manager History Data 49
Displaying Embedded Event Manager Registered Policies 50
Configuring Event SNMP Notification 51
Configuring Multiple Event Support 53
Setting the Event Configuration Parameters 53
Configuring EEM Class-Based Scheduling 55
Holding a Scheduled EEM Policy Event or Event Queue 56
Resuming Execution of EEM Policy Events or Event Queues 57
Clearing Pending EEM Policy Events or Event Queues 59
Modifying the Scheduling Parameters of EEM Policy Events or Event Queues 60
Verifying Class-Based Scheduled Activities of EEM Policies 61
Verifying Class-Based Active EEM Policies 63
Verifying Pending EEM Policies 63
Configuring EEM Applet (Interactive CLI) Support 64
Reading and Writing Input from the Active Console for Synchronous EEM Applets 64
Reading Input from the Active Console 64
Writing Input to the Active Console 66
Configuring SNMP Library Extensions 68
Prerequisites 68
SNMP Get and Set Operations 68

SNMP Get Operation 68
GetID Operation 68
SNMP Set Operation 69
SNMP Traps and Inform Requests 70
Configuring EEM Applet for SNMP Get and Set Operations 71
Configuring EEM Applet for SNMP OID Notifications 73
Configuring Variable Logic for EEM Applets 76
Prerequisites 77
Configuring Variable Logic for EEM Applets 77
Specifying a Loop of Conditional Blocks 77
Specifying if else Conditional Blocks 79
Specifying foreach Iterating Statements 80
Using Regular Expressions 82
Incrementing the Values of Variables 83
Configuring Event SNMP Object 84
Disabling AAA Authorization 86
Configuring Description of an Embedded Event Manager Applet 87
Configuration Examples for Writing EEM Policies Using the Cisco IOS CLI 88
Embedded Event Manager Applet Configuration Examples 88
Configuration Examples for Embedded Event Manager Applet 93
Example Identity Event Detector 93
Example MAT Event Detector 93
Example Neighbor-Discovery Event Detector 93
Embedded Event Manager Manual Policy Execution Examples 93
Configuring and Tracking a Stub Object Using Embedded Event Manager Example 94
Embedded Event Manager Watchdog System Monitor (Cisco IOS) Event Detector Configuration Example 94
Configuration SNMP Library Extensions Examples 95
SNMP Get Operations Examples 95
SNMP GetID Operations Examples 96
Set Operations Examples 97
Generating SNMP Notifications Examples 97
Configuring Variable Logic for EEM Applets Examples 99
Configuring Event SNMP-Object Examples 103
Configuring Description of an EEM Applet Examples 103

Additional References 103

Feature Information for Writing EEM 3.2 Policies Using the Cisco IOS CLI 105

CHAPTER 3 **Writing Embedded Event Manager Policies Using Tcl 107**

Finding Feature Information 107

Prerequisites for Writing Embedded Event Manager Policies Using Tcl 108

Information About Writing Embedded Event Manager Policies Using Tcl 108

EEM Policies 108

EEM Policy Tcl Command Extension Categories 109

General Flow of EEM Event Detection and Recovery 110

Safe-Tcl 111

Bytecode Support for EEM 2.4 113

Registration Substitution 113

Cisco File Naming Convention for EEM 114

How to Write Embedded Event Manager Policies Using Tcl 115

Registering and Defining an EEM Tcl Script 115

Displaying EEM Registered Policies 117

Unregistering EEM Policies 119

Suspending EEM Policy Execution 120

Managing EEM Policies 122

Modifying History Table Size and Displaying EEM History Data 123

Displaying Software Modularity Process Reliability Metrics Using EEM 125

Troubleshooting Tips 126

Modifying the Sample EEM Policies 126

Sample EEM Policies 126

Programming EEM Policies with Tcl 129

Tcl Policy Structure and Requirements 130

EEM Entry Status 131

EEM Exit Status 131

EEM Policies and Cisco Error Number 132

Troubleshooting Tips 139

Creating an EEM User Tcl Library Index 140

Creating an EEM User Tcl Package Index 143

Configuration Examples for Writing Embedded Event Manager Policies Using Tcl 146

Assigning a Username for a Tcl Session Examples 146

EEM Event Detector Demo Examples 147
Programming Policies with Tcl Sample Scripts Example 155
Debugging Embedded Event Manager Policies Examples 163
Tracing Tcl set Command Operations Example 165
RPC Event Detector Example 165
Additional References 167
Feature Information for Writing Embedded Event Manager 3.2 Policies Using Tcl 168

CHAPTER 4 EEM Action Tcl Command Extension 171
action_policy 172
action_process 172
action_program 174
action_reload 174
action_script 175
action_snmp_trap 176
action_snmp_object_value 176
action_switch 177
action_syslog 178
action_track_read 178
action_track_set 179

CHAPTER 5 EEM CLI Library Command Extensions 181
cli_close 182
cli_exec 182
cli_get_ttyname 183
cli_open 183
cli_read 184
cli_read_drain 185
cli_read_line 185
cli_read_pattern 186
cli_run 186
cli_run_interactive 187
cli_write 188

CHAPTER 6 EEM CLI Library XML-PI Support 193

xml_pi_exec 193
xml_pi_parse 194
xml_pi_read 195
xml_pi_write 195

CHAPTER 7 EEM Context Library Command Extensions 203
context_retrieve 203
context_save 206

CHAPTER 8 EEM Event Registration Tcl Command Extensions 211
event_register_appl 212
event_register_cli 215
event_register_counter 219
event_register_gold 221
event_register_identity 229
event_register_interface 231
event_register_ioswdsysmon 238
event_register_ipsla 242
event_register_mat 246
event_register_neighbor_discovery 247
event_register_nf 252
event_register_none 256
event_register_oir 258
event_register_process 260
event_register_resource 264
event_register_rf 266
event_register_routing 269
event_register_rpc 272
event_register_snmp 275
event_register_snmp_notification 280
event_register_snmp_object 283
event_register_syslog 286
event_register_timer 290
event_register_timer_subscriber 295
event_register_track 298

event_register_wdsysmon 300

CHAPTER 9 EEM Event Tcl Command Extensions 319

event_completion 319

event_completion_with_wait 320

event_publish 321

event_wait 324

CHAPTER 10 EEM Library Debug Command Extensions 327

cli_debug 327

smtp_debug 328

CHAPTER 11 EEM Multiple Event Support Tcl Command Extensions 329

attribute 329

correlate 330

trigger 331

CHAPTER 12 EEM SMTP Library Command Extensions 333

smtp_send_email 334

smtp_subst 335

CHAPTER 13 EEM System Information Tcl Command Extensions 337

sys_reqinfo_cli_freq 338

sys_reqinfo_cli_history 339

sys_reqinfo_cpu_all 340

sys_reqinfo_crash_history 341

sys_reqinfo_mem_all 342

sys_reqinfo_proc 344

sys_reqinfo_proc_all 345

sys_reqinfo_routername 346

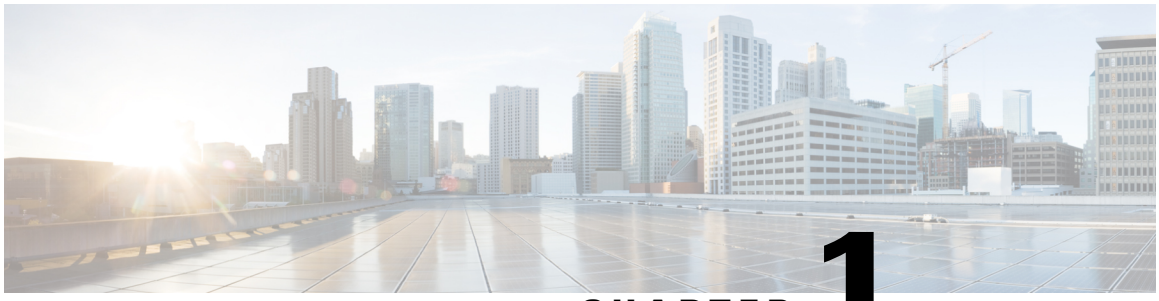
sys_reqinfo_snmp 346

sys_reqinfo_syslog_freq 347

sys_reqinfo_syslog_history 348

CHAPTER 14 EEM Utility Tcl Command Extensions 351

appl_read 352
appl_reqinfo 353
appl_setinfo 353
counter_modify 354
description 355
fts_get_stamp 356
register_counter 357
register_timer 358
timer_arm 359
timer_cancel 361
unregister_counter 362



CHAPTER 1

Embedded Event Manager Overview

Embedded Event Manager (EEM) is a distributed and customized approach to event detection and recovery offered directly in a Cisco IOS device. EEM offers the ability to monitor events and take informational, corrective, or any desired EEM action when the monitored events occur or when a threshold is reached. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs.

This module contains a technical overview of EEM. EEM can be used alone, or with other network management technologies to help monitor and maintain your network. Before you begin to implement EEM, it is important that you understand the information presented in this module.

- Finding Feature Information, page 1
- Information About Embedded Event Manager, page 1
- Where to Go Next, page 20
- Additional References, page 20
- Feature Information for Embedded Event Manager 3.2 Overview, page 21

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

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

Information About Embedded Event Manager

Embedded Event Manager

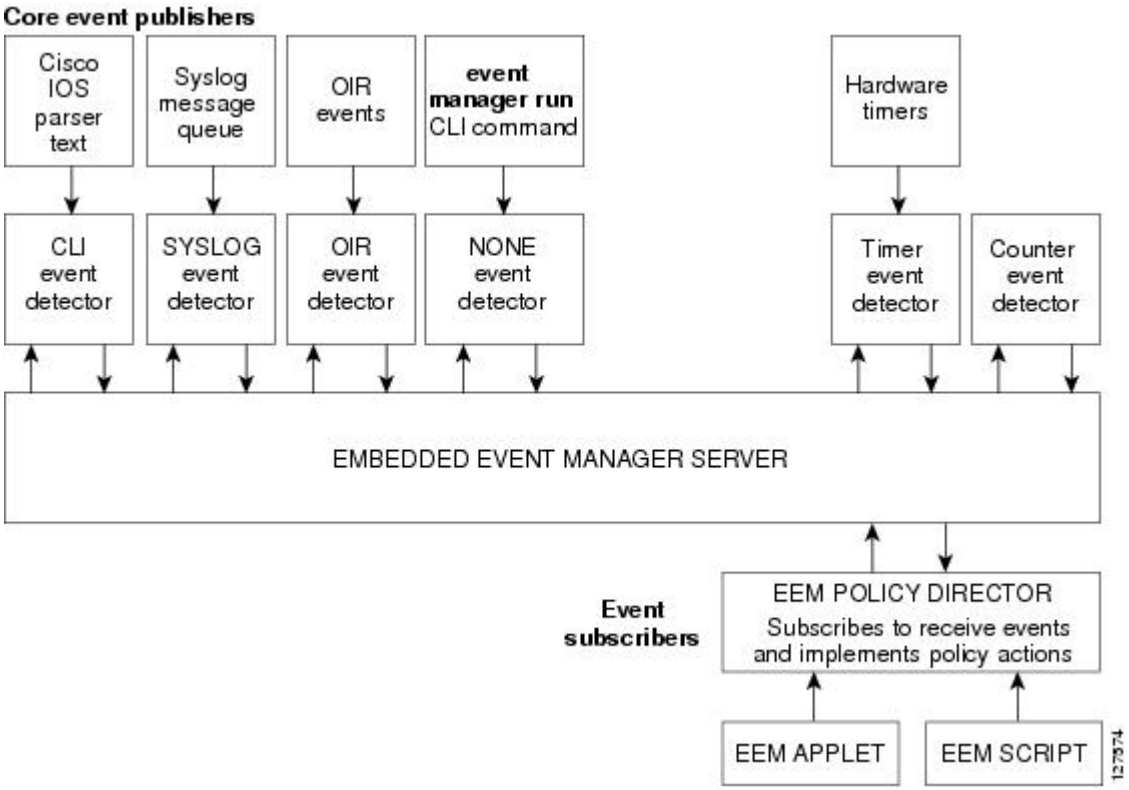
Event tracking and management has traditionally been performed by devices external to the networking device. Embedded Event Manager (EEM) has been designed to offer event management capability directly in Cisco

IOS devices. The on-device, proactive event management capabilities of EEM are useful because not all event management can be done off device because some problems compromise communication between the device and the external network management device. Capturing the state of the device during such situations can be invaluable in taking immediate recovery actions and gathering information to perform root-cause analysis. Network availability is also improved if automatic recovery actions are performed without the need to fully reboot the routing device.

EEM is a flexible, policy-driven framework that supports in-box monitoring of different components of the system with the help of software agents known as event detectors. The figure below shows the relationship between the EEM server, core event publishers (event detectors), and the event subscribers (policies). Basically, event publishers screen events and publish them when there is a match on an event specification that is provided by the event subscriber. Event detectors notify the EEM server when an event of interest occurs. The EEM policies that are configured using the Cisco command-line interface (CLI) then implement recovery on the basis of the current state of the system and the actions specified in the policy for the given event.

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

Figure 1: Embedded Event Manager Core Event Detectors



Note If your network has a higher version of EEM, that version would include the previous releases of EEM version as well.

Embedded Event Manager 1.0

EEM 1.0 introduced Embedded Event Manager. EEM 1.0 introduced the following event detectors:

- SNMP—The Simple Network Management Protocol (SNMP) event detector allows a standard SNMP MIB object to be monitored and an event to be generated when the object matches specified values or crosses specified thresholds.
- Syslog—The syslog event detector allows for screening syslog messages for a regular expression pattern match.

EEM 1.0 introduced the following actions:

- Generating prioritized syslog messages.
- Generating a Cisco Networking Services (CNS) event for upstream processing by CNS devices.
- Reloading the Cisco software.
- Switching to a secondary processor in a fully redundant hardware configuration.

Embedded Event Manager 2.0

EEM 2.0 introduced some new features. EEM 2.0 introduced the following event detectors:

- Application-Specific—The application-specific event detector allows any Embedded Event Manager policy to publish an event.
- Counter—The counter event detector publishes an event when a named counter crosses a specified threshold.
- Interface Counter—The interface counter event detector publishes an event when a generic Cisco IOS interface counter for a specified interface crosses a defined threshold.
- Timer—The timer event detector publishes events for the following four different types of timers: absolute-time-of-day, countdown, watchdog, and CRON.
- Watchdog System Monitor (IOSWDSysMon)—The Cisco IOS watchdog system monitor event detector publishes an event when CPU or memory utilization for a Cisco IOS process crosses a threshold.

EEM 2.0 introduced the following actions:

- Setting or modifying a named counter.
- Publishing an application-specific event
- Generating an SNMP trap.

The ability to run a Cisco defined sample policy written using Tool Command Language (Tcl) was introduced. A sample policy was provided that could be stored in the system policy directory.

Embedded Event Manager 2.1

EEM 2.1 and introduced some new features. EEM 2.1 introduced the following new event detectors:

- CLI—The CLI event detector screens command-line interface (CLI) commands for a regular expression match.
- None—The none event detector publishes an event when the Cisco IOS **event manager run** command executes an EEM policy.
- OIR—The online insertion and removal (OIR) event detector publishes an event when a particular hardware insertion or removal event occurs.

EEM 2.1 introduced the following actions:

- Executing a Cisco CLI command.
- Requesting system information when an event occurs.
- Sending a short e-mail.
- Manually running an EEM policy.

EEM 2.1 also permits multiple concurrent policies to be run using the new **event manager scheduler script** command. Support for SNMP event detector rate-based events is provided as is the ability to create policies using Tool Command Language (Tcl).

Embedded Event Manager 2.1 (Software Modularity)

EEM 2.1 (Software Modularity) is supported on Cisco Software Modularity images. EEM 2.1 (Software Modularity) introduced the following event detectors:

- GOLD—The Generic Online Diagnostic (GOLD) event detector publishes an event when a GOLD failure event is detected on a specified card and subcard.
- System Manager—The system manager event detector generates events for Cisco IOS Software Modularity process start, normal or abnormal stop, and restart events. The events generated by the system manager allows policies to change the default behavior of the process restart.
- Watchdog System Monitor (WDSysMon)—The Cisco Software Modularity watchdog system monitor event detector detects infinite loops, deadlocks, and memory leaks in Cisco IOS Software Modularity processes.

EEM 2.1 for Software Modularity introduced the ability to display EEM reliability metric data for processes.

Note EEM 2.1 for Software Modularity images also supports the resource and RF event detectors introduced in EEM 2.2, but it does not support the enhanced object tracking event detector or the actions to read and set tracked objects.

Embedded Event Manager 2.2

EEM 2.2 introduced some new features. EEM 2.2 introduced the following event detectors:

- Enhanced Object Tracking—The enhanced object tracking event detector publishes an event when the tracked object changes. Enhanced object tracking provides complete separation between the objects to be tracked and the action to be taken by a client when a tracked object changes.
- Resource—The resource event detector publishes an event when the Embedded Resource Manager (ERM) reports an event for the specified policy.
- RF—The redundancy framework (RF) event detector publishes an event when one or more RF events occur during synchronization in a dual Route Processor (RP) system. The RF event detector can also detect an event when a dual RP system continuously switches from one RP to another RP (referred to as a ping-pong situation).

EEM 2.2 introduced the following actions:

- Reading the state of a tracked object.
- Setting the state of a tracked object.

Embedded Event Manager 2.3

EEM 2.3 is supported on the Cisco Catalyst 6500 Series switches and introduces enhancements to the Generic Online Diagnostics (GOLD) Event Detector on that product.

- The **event gold** command was enhanced with the addition of the **action-notify**, **testing-type**, **test-name**, **test-id**, **consecutive-failure**, **platform-action**, and **maxrun** keywords for improved reaction to GOLD test failures and conditions.
- The following platform-wide GOLD Event Detector information can be accessed through new read-only EEM built-in environment variables:
  - Boot-up diagnostic level
  - Card index, name, serial number
  - Port counts
  - Test counts
- The following test-specific GOLD Event Detector information can be accessed through new read-only EEM built-in environment variables (available to EEM applets only):
  - Test name, attribute, total run count
  - Test result per test, port, or device
  - Total failure count, last fail time
  - Error code
  - Occurrence of consecutive failures

These enhancements result in reduced mean time to recovery (MTTR) and higher availability through improved automation and fault detection.

Embedded Event Manager 2.4

EEM 2.4 introduced the following event detectors:

- SNMP Notification—The SNMP notification event detector provides the ability to intercept SNMP trap and inform messages coming into the device. An SNMP notification event is generated when an incoming SNMP trap or inform message matches specified values or crosses specified thresholds.
- RPC—The remote procedure call (RPC) event detector provides the ability to invoke EEM policies from outside the device over an encrypted connection using Secure Shell (SSH). The RPC event detector uses Simple Object Access Protocol (SOAP) data encoding for exchanging XML-based messages. This event detector can be used to run EEM policies and then receive output in a SOAP XML-formatted reply.

EEM 2.4 added enhancements to the following event detectors:

- Interface counter rate-based trigger—This feature adds the ability for an interface event to be triggered based on a rate of change over a period of time. A rate can be specified both for the entry value and the exit value. This feature copies the rate-based functionality that currently exists for the SNMP event detector.
- SNMP delta value—The difference between the monitored Object Identifier (OID) value at the beginning of the monitored period and the actual OID value when the event is published will be provided in the **event reqinfo** data for both the SNMP event detector and the Interface Counter event detector.

EEM 2.4 introduced the following actions:

- Multiple event support—The ability to run multiple events was introduced. and **show event manager** commands were enhanced to show multiple events.
- Support for parameters—The *parameter* argument has been added to the **event manager run** command. A maximum of 15 parameters can be used.
- Display of Job IDs and completion status--Some of the **show event manager** commands were enhanced to display Job IDs and completion status.
- Bytecode support—Tcl 8 defines a specialized bytecode language (BCL) and includes a just-in-time compiler that translates Tcl scripts to BCL. Byte sequence is executed by a “virtual machine,” Tcl_ExecuteByteCode(), or TEBC for short, as often as needed. Currently EEM accepts file extensions, such as *.tcl for user policies and *.tm for system policies. Tcl standard extension for bytecode scripts are *.tbc. Now EEM will accept *.tbc as valid EEM policies.
- Registration substitution enhancement—Supports replacing multiple parameters in the event registration statement lines with a single environment variable.
- Tcl package support

Embedded Event Manager 3.0

EEM 3.0 introduces the following new event detectors:

- Custom CLI--The custom CLI event detector publishes an event to add and enhance existing CLI command syntax.
- Routing--The Routing event detector publishes an event when route entries change in the Routing Information Base (RIB).
- NetFlow-- The NetFlow event detector publishes an event when a NetFlow event is triggered.
- IP SLA--The IP SLA event detector publishes an event when an IP SLA reaction is triggered.

EEM 3.0 introduces the following features.

- Class-based scheduling--The EEM policies will be assigned a class using the **class** keyword when they are registered. EEM policies registered without a class will be assigned to the default class.
- High performance Tcl policies--Three new Tcl commands are introduced **event_completion**, **event_wait**, and **event_completion_with_wait**.
- Interactive cli support--The synchronous applets are enhanced to support interaction with the local console (TTY). Two new IOS commands, **action gets** and **action puts,** are introduced to allow users to enter and display input directly on the console.
- Variable logic for applets--The Variable Logic for EEM Applets feature adds the ability to apply conditional logic within EEM applets. Conditional logic introduces a control structure that can change the flow of actions within applets depending on conditional expressions.
- Digital signature support--A new API performs digital signature verification for a Tcl script to check it the script is signed by Cisco before execution.
- Support authenticating e-mail servers--The **action mail**command is modified to include an optional username and password.
- SMTP IPv6 support--The keyword **sourceaddr** is added in Tcl e-mail templates to specify either an IPv6 or IPv4 address.
- SNMP library extensions--The EEM applet **action info** and Tcl **sys_reqinfo_snmp**commands are enhanced to include functionality for SNMP getid, inform, trap, and set-type operations.
- SNMP Notification IPv6 support--IPv6 address is supported for the source and destination IP addresses.
- CLI Library XML-PI support--Provides a programmable interface which encapsulates IOS command-line interface (CLI) show commands in XML format in a consistent way across different Cisco products. Customers using XML-PI will be able to parse IOS show command output from within Tcl scripts using well-known keywords instead of having to depend on the use of regular expression support.

Embedded Event Manager 3.1

EEM 3.1 introduced one new event detector:

- SNMP Object--The Simple Network Management Protocol (SNMP) object trap event detector provides an extension to replace the value when an SNMP trap with the specified SNMP object ID (OID) is encountered on a specific interface or address.

EEM 3.1 added an enhancement to the following event detector:

- SNMP Notification--The SNMP notification event detector now can wait and intercept the outgoing SNMP traps and informs.

EEM 3.1 added enhancement to the following action:

- Specify facility--The **action syslog** command has been enhanced to specify syslog facility.

EEM 3.1 introduces the following features:

- Provides the ability to create a short description for the registered policy--A new **description** command has been introduced to register policies with a brief description in Cisco IOS CLI and Tcl policies. The **show event manager policy available** command and the **show event manager policy registered** command have been enhanced to add the **description** keyword to display the description of the registered applet.
- Enables EEM policies to bypass AAA authorization--The **event manager application** command has been enhanced to provide authorization and bypass keywords to disable AAA.
- Introduces CLI Library enhancements--Provides two new commands in the CLI library: **cli_run** and **cli_run_interactive**.

Embedded Event Manager 3.2

EEM 3.2 introduced the following new event detectors:

- Neighbor Discovery--Neighbor Discovery event detector provides the ability to publish a policy to respond to automatic neighbor detection when:
  - a Cisco Discovery Protocol (CDP) cache entry is added, deleted or updated.
  - a Link Layer Discovery Protocol (LLDP) cache entry is added, deleted, or updated.
  - an interface link status changes.
  - an interface line status changes.
- Identity--Identity event detector generates an event when AAA authorization and authentication is successful, when failure occurs, or after normal user traffic on the port is allowed to flow.
- Mac-Address-Table--Mac-Address-Table event detector generates an event when a MAC address is learned in the MAC address table.

Note The Mac-Address-Table event detector is supported only on switch platforms and can be used only on Layer 2 interfaces where MAC addresses are learned. Layer 3 interfaces do not learn addresses and devices do not usually support the mac-address-table infrastructure needed to notify EEM of a learned MAC address.

EEM 3.2 also introduces new CLI commands to support the applets to work with the new event detectors.

EEM Event Detectors Available by Cisco IOS Release

EEM uses software programs known as event detectors to determine when an EEM event occurs. Some event detectors are available on every Cisco IOS release, but most event detectors have been introduced in a specific

release. Use the table below to determine which event detectors are available in your specific Cisco IOS release. A blank entry (--) indicates that the event detector is not available: the text "Yes" indicates that the event detector is available. The event detectors shown in the table are supported in later releases of the same Cisco IOS release train. For more details on each event detector, see the Event Detectors concept in the "Embedded Event Manager Overview" module.

Table 1: Availability of Event Detectors by Cisco IOS Release

Event Detector	12.2(25)S	12.3(14)T 12.2(18)SXF5 12.2(28)SB 12.2(33)SRA	12.4(2)T 12.2(31)SB3 12.2(33)SRB	12.2(18)SXF4 Cisco IOS Software Modularity	12.2(33)SXH	12.4(20)T 12.2(33)SXI	12.4(22)T 12.2(33)SRE	15.0(1)M 15.1(3)T	15.1(2)SY	15 E XE 3E
Application Specific	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CLI	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	--	Yes
Counter	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Custom CLI	--	--	--	--	--	--	Yes	Yes	--	--
Enhanced Object Tracking	--	--	Yes	--	Yes	Yes	Yes	Yes	--	--
Environmental	--	--	--	--	--	--	--	--	--	Yes
GOLD	--	--	--	Yes	Yes	Yes	Yes	Yes	--	Yes
Identity	--	--	--	--	--	--	--	Yes	Yes	Yes
Interface Counter	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	--	Yes
IPSLA	--	--	--	--	--	--	Yes	Yes	--	Yes
Media Flow	--	--	--	--	--	--	--	Yes	Yes	Yes
Neighbor Discovery	--	--	--	--	--	--	--	Yes	Yes	Yes
NF	--	--	--	--	--	--	Yes	Yes	--	--
None	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
OIR	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Resource	--	--	Yes	Yes	Yes	Yes	Yes	Yes	--	--

Event Detector	12.2(25)S	12.3(14)T 12.2(18)SXF5 12.2(28)SB 12.2(33)SRA	12.4(2)T 12.2(31)SB3 12.2(33)SRB	12.2(18)SXF4 Cisco IOS Software Modularity	12.2(33)SXH	12.4(20)T 12.2(33)SXI	12.4(22)T 12.2(33)SRE	15.0(1)M 15.1(3)T	15.1(2)SY	15 E XE 3E
RF	--	--	Yes	Yes	Yes	Yes	Yes	Yes	--	Yes
Routing	--	--	--	--	--	--	Yes	Yes	--	Yes
RPC	--	--	--	--	--	Yes	Yes	Yes	Yes	--
SNMP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	--	Yes
SNMP Proxy	--	--	--	--	--	--	--	--	Yes	--
SNMP Notification	--	--	--	--	--	Yes	Yes	Yes	--	Yes
SNMP Object	--	--	--	--	--	--	--	Yes	--	Yes
Syslog	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
System Manager	--	--	--	Yes	Yes	Yes	Yes	Yes	Yes	--
Timer	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IOSWDSysMon (Cisco IOS watchdog)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	--	Yes
WDSysMon (Cisco IOS Software Modularity watchdog)	--	--	--	Yes	--	--	--		--	--

Event Detectors

Embedded Event Manager (EEM) uses software programs known as *event detectors* to determine when an EEM event occurs. Event detectors are separate systems that provide an interface between the agent being monitored, for example Simple Network Management Protocol (SNMP), and the EEM policies where an action can be implemented. Some event detectors are available on every Cisco IOS release, but most event

detectors have been introduced in a specific release. For details of which event detector is supported in each Cisco IOS release, see the EEM Event Detectors Available by Cisco IOS Release concept in the “Writing Embedded Event Manager Policies Using the Cisco IOS CLI” or the “Writing Embedded Event Manager Policies Using Tcl” modules. EEM contains the following event detectors.

Application-Specific Event Detector

The application-specific event detector allows any Embedded Event Manager policy to publish an event. When an EEM policy publishes an event it must use an EEM subsystem number of 798 with any event type. If an existing policy is registered for subsystem 798 and a specified event type, a second policy of the same event type will trigger the first policy to run when the specified event is published.

CLI Event Detector

The CLI event detector screens command-line interface (CLI) commands for a regular expression match. When a match is found, an event is published. The match logic is performed on the fully expanded CLI command after the command is successfully parsed and before it is executed. The CLI event detector supports three publish modes:

- Synchronous publishing of CLI events--The CLI command is not executed until the EEM policy exits, and the EEM policy can control whether the command is executed. The read/write variable, _exit_status, allows you to set the exit status at policy exit for policies triggered from synchronous events. If _exit_status is 0, the command is skipped, if _exit_status is 1, the command is run.
- Asynchronous publishing of CLI events--The CLI event is published, and then the CLI command is executed.
- Asynchronous publishing of CLI events with command skipping--The CLI event is published, but the CLI command is not executed.

Counter Event Detector

The counter event detector publishes an event when a named counter crosses a specified threshold. There are two or more participants that affect counter processing. The counter event detector can modify the counter, and one or more subscribers define the criteria that cause the event to be published. After a counter event has been published, the counter monitoring logic can be reset to start monitoring the counter immediately or it can be reset when a second threshold--called an exit value--is crossed.

Custom CLI Event Detector

The custom CLI event detector publishes an event to add and enhance existing CLI command syntax. When the special parser characters Tab, ? (question mark), and Enter are entered, the parser sends the input to the custom CLI event detector for processing. The custom CLI event detector then compares this input against registered strings to determine if this is a new or enhanced CLI command. Upon a match the custom CLI event detector takes appropriate actions, such as displaying help for the command if ? is entered, displaying the entire command if Tab is entered, or executing the command if Enter was entered. If a match does not occur, the parser regains control and processes the information as usual.

Enhanced Object Tracking Event Detector

The enhanced object tracking (EOT) event detector publishes an event when the status of a tracked object changes. Object tracking was first introduced into the Hot Standby Router Protocol (HSRP) as a simple tracking mechanism that allowed you to track the interface line-protocol state only. If the line-protocol state

of the interface went down, the HSRP priority of the device was reduced, allowing another HSRP device with a higher priority to become active.

Object tracking was enhanced to provide complete separation between the objects to be tracked and the action to be taken by a client when a tracked object changes. Thus, several clients such as HSRP, VRRP, or GLBP can register their interest with the tracking process, track the same object, and each take different action when the object changes. Each tracked object is identified by a unique number that is specified on the tracking command-line interface (CLI). Client processes use this number to track a specific object. The tracking process periodically polls the tracked objects and notes any change of value. The changes in the tracked object are communicated to interested client processes, either immediately or after a specified delay. The object values are reported as either up or down.

Enhanced object tracking is now integrated with EEM to allow EEM to report on a status change of a tracked object and to allow enhanced object tracking to track EEM objects. A new type of tracking object--a stub object--is created. The stub object can be manipulated using the existing CLI commands that already allow tracked objects to be manipulated.

GOLD Event Detector

The GOLD event detector publishes an event when a GOLD failure event is detected on a specified card and subcard.

Interface Counter Event Detector

The interface counter event detector publishes an event when a generic Cisco IOS interface counter for a specified interface crosses a defined threshold. A threshold can be specified as an absolute value or an incremental value. If the incremental value is set to 50, for example, an event would be published when the interface counter increases by 50.

After an interface counter event has been published, the interface counter monitoring logic is reset using two methods. The interface counter is reset either when a second threshold--called an exit value--is crossed or when an elapsed period of time occurs.

IP SLA Event Detector

The IP SLA event detector publishes an event when an IP SLA reaction is triggered.

NetFlow Event Detector

The NetFlow event detector publishes an event when a NetFlow event is triggered.

None Event Detector

The none event detector publishes an event when the Cisco IOS **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.

OIR Event Detector

The online insertion and removal (OIR) event detector publishes an event when one of the following hardware insertion or removal events occurs:

- A card is removed.
- A card is inserted.

Route Processors (RPs), line cards, or feature cards can be monitored for OIR events.

Resource Event Detector

The resource event detector publishes an event when the Embedded Resource Manager (ERM) reports an event for the specified policy. The ERM infrastructure tracks resource depletion and resource dependencies across processes and within a system to handle various error conditions. The error conditions are handled by providing an equitable sharing of resources between various applications. The ERM framework provides a communication mechanism for resource entities and allows communication between these resource entities from numerous locations. The ERM framework also helps in debugging CPU and memory-related issues. The ERM monitors system resource usage to better understand scalability needs by allowing you to configure threshold values for resources such as the CPU, buffers, and memory. The ERM event detector is the preferred method for monitoring resources in Cisco software but the ERM event detector is not supported in Software Modularity images. For more details about ERM, go to “Embedded Resource Manager” module.

RF Event Detector

The redundancy framework (RF) event detector publishes an event when one or more RF events occur during synchronization in a dual Route Processor (RP) system. The RF event detector can also detect an event when a dual RP system continuously switches from one RP to another RP (referred to as a ping-pong situation).

RPC Event Detector

The remote procedure call (RPC) event detector provides the ability to invoke EEM policies from outside the device over an encrypted connection using Secure Shell (SSH). The RPC event detector uses Simple Object Access Protocol (SOAP) data encoding for exchanging XML-based messages. This event detector can be used to run EEM policies and then receive output in a SOAP XML-formatted reply.

Routing Event Detector

The routing event detector publishes an event when a route entry changes in the Routing Information Base (RIB).

SNMP Event Detector

The SNMP event detector allows a standard SNMP MIB object to be monitored and an event to be generated when the object matches specified values or crosses specified thresholds.

SNMP Notification Event Detector

The SNMP notification event detector provides the ability to intercept SNMP trap and inform messages coming into or going out of the device. An SNMP notification event is generated when an incoming or outgoing SNMP trap or inform message matches specified values or crosses specified thresholds. The SNMP event detector can wait and intercept the outgoing SNMP traps and informs.

SNMP Object Event Detector

The Simple Network Management Protocol (SNMP) object trap event detector provides an extension to replace the value when an SNMP trap with the specified SNMP object ID (OID) is encountered on a specific interface or address.

Syslog Event Detector

The syslog event detector allows for screening syslog messages for a regular expression pattern match. The selected messages can be further qualified, requiring that a specific number of occurrences be logged within a specified time. A match on a specified event criteria triggers a configured policy action.

System Manager Event Detector

The system manager event detector generates events for Cisco IOS Software Modularity process start, normal or abnormal stop, and restart events. The events generated by the system manager allows policies to change the default behavior of the process restart.

Timer Event Detector

The timer event detector publishes events for the following four different types of timers:

- An absolute-time-of-day timer publishes an event when a specified absolute date and time occurs.
- A countdown timer publishes an event when a timer counts down to zero.
- A watchdog timer publishes an event when a timer counts down to zero and then the timer automatically resets itself to its initial value and starts to count down again.
- A CRON timer publishes an event using a UNIX standard CRON specification to indicate when the event is to be published. A CRON timer never publishes events more than once per minute.

Watchdog System Monitor (IOSWDSysMon) Event Detector for Cisco IOS

The Cisco IOS watchdog system monitor event detector publishes an event when one of the following occurs:

- CPU utilization for a Cisco IOS task crosses a threshold.
- Memory utilization for a Cisco IOS task crosses a threshold.

Note Cisco IOS processes are now referred to as tasks to distinguish them from Cisco IOS 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.

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

The Cisco IOS Software Modularity watchdog system monitor event detector detects infinite loops, deadlocks, and memory leaks in Cisco IOS Software Modularity processes.

EEM Actions Available by Cisco IOS Release

The CLI-based corrective actions that are taken when event detectors report events enable a powerful on-device event management mechanism. Some actions are available in every Cisco IOS release, but most actions have been introduced in a specific release. Use the table below to determine which actions are available in your specific Cisco IOS release. A blank entry (--) indicates that the action is not available; the text “Yes” indicates that the action is available. The actions shown in the table are supported in later releases of the same Cisco

IOS release train. For more details on each action, see the Embedded Event Manager Actions concept in the "Embedded Event Manager Overview" module.

Table 2: Availability of Actions by Cisco IOS Release

Action	12.2(25)S	12.3(14)T 12.2(18)SXF5 12.2(28)SB 12.2(33)SRA	12.4(2)T 12.2(31)SB3 12.2(33)SRB	12.2(18)SXF4 Cisco IOS Software Modularity	12.2(33)SXH	12.4(20)T	12.4(22)T	15.0(1)M	15E XE 3E
Execute a CLI command	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Generate a CNS event	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Generate a prioritized syslog message	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Generate an SNMP trap	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Manually run an EEM policy	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Publish an application-specific event	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Read the state of a tracked object	--	--	Yes	--		Yes	Yes	Yes	Yes
Reload the Cisco software	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Request system information	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Send a short e-mail	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Action	12.2(25)S	12.3(14)T 12.2(18)SXF5 12.2(28)SB 12.2(33)SRA	12.4(2)T 12.2(31)SB3 12.2(33)SRB	12.2(18)SXF4 Cisco IOS Software Modularity	12.2(33)SXH	12.4(20)T	12.4(22)T	15.0(1)M	15E XE 3E
Set or modify a named counter	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Set the state of a tracked object	--	--	Yes	--		Yes	Yes	Yes	Yes
Switch to a secondary RP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Embedded Event Manager Actions

The CLI-based corrective actions that are taken when event detectors report events enable a powerful on-device event management mechanism. Some EEM actions are available on every Cisco IOS release, but most EEM actions have been introduced in a specific release. For details of which EEM action is supported in each Cisco IOS release, see the EEM Actions Available by Cisco IOS Release concept in the “Writing Embedded Event Manager Policies Using the Cisco IOS CLI” or the “Writing Embedded Event Manager Policies Using Tcl” modules. EEM supports the following actions:

- Executing a Cisco IOS command-line interface (CLI) command.
- Generating a CNS event for upstream processing by Cisco CNS devices.
- Setting or modifying a named counter.
- Switching to a secondary processor in a fully redundant hardware configuration.
- Requesting system information when an event occurs.
- Sending a short e-mail.
- Manually running an EEM policy.
- Publishing an application-specific event.
- Reloading the Cisco software.
- Generating an SNMP trap.
- Generating prioritized syslog messages.
- Reading the state of a tracked object.
- Setting the state of a tracked object.

EEM action CLI commands contain an EEM action label that is a unique identifier that can be any string value. Actions are sorted and run in ascending alphanumeric (lexicographical) key sequence using the label as the sort key. If you are using numbers as labels be aware that alphanumerical sorting will sort 10.0 after 1.0, but before 2.0, and in this situation we recommend that you use numbers such as 01.0, 02.0, and so on, or use an initial letter followed by numbers.

Embedded Event Manager Environment Variables

EEM allows environment variables to be used in EEM policies. Tool Command Language (Tcl) allows global variables to be defined that are known to all procedures within a Tcl script. EEM allows environment variables to be defined using a CLI command, the **event manager environment** command, for use within an EEM policy. All EEM environment variables are automatically assigned to Tcl global variables before a Tcl script is run. There are three different types of environment variables associated with Embedded Event Manager:

- User-defined--Defined by you if you create an environment variable in a policy that you have written.
- Cisco-defined--Defined by Cisco for a specific sample policy.
- Cisco built-in (available in EEM applets)--Defined by Cisco and can be read only or read/write. The read only variables are set by the system before an applet starts to execute. The single read/write variable, _exit_status, allows you to set the exit status at policy exit for policies triggered from synchronous events.

Cisco-defined environment variables (see the table below) and Cisco system-defined environment variables may apply to one specific event detector or to all event detectors. Environment variables that are user-defined or defined by Cisco in a sample policy are set using the **event manager environment** command. Variables that are used in the EEM policy must be defined before you register the policy. A Tcl policy contains a section called “Environment Must Define” that can be defined to check that any required environment variables are defined before the policy runs.

Cisco built-in environment variables are a subset of the Cisco-defined environment variables and the built-in variables are available to EEM applets only. The built-in variables can be read-only or can be read and write, and these variables may apply to one specific event detector or to all event detectors. For more details and a table listing the Cisco system-defined variables, see the ““Writing Embedded Event Manager Policies Using the Cisco IOS CLI” module.

Note Cisco-defined environment variables begin with an underscore character (_). We strongly recommend that customers avoid the same naming convention to prevent naming conflicts.

The table below describes the Cisco-defined variables used in the sample EEM policies. Some of the environment variables do not have to be specified for the corresponding sample policy to run and these are marked as optional.

Table 3: Cisco-Defined Environmental Variables and Examples

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

Environment Variable	Description	Example
_config_cmd2	(Optional) The second configuration command that is executed.	**no shutdown**
_crash_reporter_debug	(Optional) A value that identifies whether debug information for tm_crash_reporter.tcl will be enabled.	1
_crash_reporter_url	The URL location to which the crash report is sent.	http://www.yourdomain.com/fm/interface_tm.cgi
_cron_entry	A CRON specification that determines when the policy will run. See the "Writing Embedded Event Manager Policies Using Tcl" module for more information about how to specify a cron entry.	0-59/1 0-23/1 * * 0-7
_email_server	A Simple Mail Transfer Protocol (SMTP) mail server used to send e-mail.	mailserver.yourdomain.com
_email_to	The address to which e-mail is sent.	engineer@yourdomain.com
_email_from	The address from which e-mail is sent.	devtest@yourdomain.com
_email_cc	The address to which the e-mail is be copied.	manager@yourdomain.com
_email_ipaddr	The source IP address of the recipient.	209.165.201.1 or (IPv6 address) 2001:0DB8::1
_info_snmp_oid	The SNMP object ID.	1.3.6.1.2.1.2 or iso.internet.mgmt.mib-2.interfaces
_info_snmp_value	The value string of the associated SNMP data element.	
_show_cmd	The CLI **show** command to be executed when the policy is run.	**show version**
_syslog_pattern	A regular expression pattern match string that is used to compare syslog messages to determine when the policy runs.	.*UPDOWN.*FastEthernet 0/0.*

Environment Variable	Description	Example
_tm_fsys_usage_cron	(Optional) A CRON specification that is used in the **event_register** keyword extension. If unspecified, the _tm_fsys_usage.tcl policy is triggered once per minute.	0-59/1 0-23/1 * * 0-7
_tm_fsys_usage_debug	(Optional) When this variable is set to a value of 1, disk usage information is displayed for all entries in the system.	1
_tm_fsys_usage_ freebytes	(Optional) Free byte threshold for systems or specific prefixes. If free space falls below a given value, a warning is displayed.	disk2:98000000
_tm_fsys_usage_percent	(Optional) Disk usage percentage thresholds for systems or specific prefixes. If disk usage percentage exceeds a given percentage, a warning is displayed. If unspecified, the default disk usage percentage is 80 percent for all systems.	nvram:25 disk2:5

Embedded Event Manager Policy Creation

EEM is a policy driven process in which the EEM policy engine receives notifications when faults and other events occur in the Cisco software system. Embedded Event Manager policies implement recovery based on the current state of the system and the actions specified in the policy for a given event. Recovery actions are triggered when the policy is run.

Although there are some EEM CLI configuration and **show** commands, EEM is implemented through the creation of policies. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs. There are two types of EEM policies: an applet or a script. An applet is a simple form of policy that is defined within the CLI configuration. A script is a form of policy that is written in Tcl.

The creation of an EEM policy involves:

- Selecting the event for which the policy is run.
- Defining the event detector options associated with logging and responding to the event.
- Defining the environment variables, if required.
- Choosing the actions to be performed when the event occurs.

There are two ways to create an EEM policy. The first method is to write applets using CLI commands, and the second method is to write Tcl scripts. Cisco provides enhancements to Tcl in the form of Tcl command

extensions that facilitate the development of EEM policies. Scripts are defined off the networking device using an ASCII editor. The script is then copied to the networking device and registered with EEM. When a policy is registered with the Embedded Event Manager, the software examines the policy and registers it to be run when the specified event occurs. Policies can be unregistered or suspended. Both types of policies can be used to implement EEM in your network.

For details on writing EEM policies using the Cisco IOS CLI, go to "Writing Embedded Event Manager Policies Using the Cisco IOS CLI" module.

For details on writing EEM policies using Tcl, go to "Writing Embedded Event Manager Policies Using Tcl" module.

Where to Go Next

- If you want to write EEM policies using the Cisco IOS CLI, see the "Writing Embedded Event Manager Policies Using the Cisco IOS CLI" module.
- If you want to write EEM policies using Tcl, see the "Writing Embedded Event Manager Policies Using Tcl" module.

Additional References

The following sections provide references related to EEM.

Related Documents

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

Standards

Standard	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--

MIBs

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

RFCs

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

Technical Assistance

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

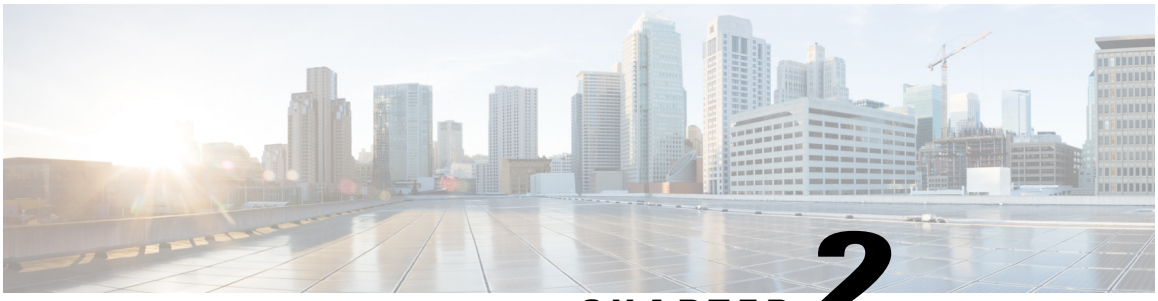
Feature Information for Embedded Event Manager 3.2 Overview

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

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

Table 4: Feature Information for Embedded Event Manager 3.2 Overview

Feature Name	Releases	Feature Information
Embedded Event Manager 3.2	Cisco IOS XE Release 3.6E	EEM 3.2 introduced several new features. The following commands were introduced or modified: **debug event manager**, **event identity**, **event mat**, **event neighbor-discovery**, **show event manager detector**.



CHAPTER 2

Writing Embedded Event Manager Policies Using the Cisco IOS CLI

This module describes how to write Embedded Event Manager (EEM) policies using Cisco IOS command-line interface (CLI) applets to handle Cisco software faults and events. EEM is a distributed and customized approach to event detection and recovery offered directly in a Cisco IOS device. EEM offers the ability to monitor events and take informational, corrective, or any desired action when the monitored events occur or when a threshold is reached. The EEM policy engine receives notifications when faults and other events occur. EEM policies implement recovery on the basis of the current state of the system and the actions specified in the policy for a given event. Recovery actions are triggered when the policy is run.

- Finding Feature Information, page 23
- Prerequisites for Writing EEM Policies Using the Cisco IOS CLI, page 24
- Information About Writing EEM Policies Using the Cisco IOS CLI, page 24
- How to Write EEM Policies Using the Cisco IOS CLI, page 37
- Configuration Examples for Writing EEM Policies Using the Cisco IOS CLI, page 88
- Additional References, page 103
- Feature Information for Writing EEM 3.2 Policies Using the Cisco IOS CLI, page 105

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

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

Prerequisites for Writing EEM Policies Using the Cisco IOS CLI

- Before writing EEM policies, you should be familiar with the concepts explained in the "Embedded Event Manager Overview" module.
- If the **action cns-event** command is used, access to a Cisco Networking Services (CNS) Event gateway must be configured.
- If the **action force-switchover** command is used, a secondary processor must be configured on the device.
- If the **action snmp-trap** command is used, the **snmp-server enable traps event-manager** command must be enabled to permit SNMP traps to be sent from the Cisco IOS device to the SNMP server. Other relevant **snmp-server** commands must also be configured; for details see the **action snmp-trap** command page.

Information About Writing EEM Policies Using the Cisco IOS CLI

Embedded Event Manager Policies

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

EEM Applet

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

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

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

The action configuration commands are uniquely identified using the *label* argument, which can be any string value. Actions are sorted in ascending alphanumeric key sequence using the *label* argument as the sort key, and they are run using this sequence.

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

EEM Script

Scripts are defined off the networking device using an ASCII editor. The script is then copied to the networking device and registered with EEM. Tcl scripts are supported by EEM.

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

- Selecting the event for which the policy is run.
- Defining the event detector options associated with logging and responding to the event.
- Choosing the actions to be followed when the event occurs.

Cisco provides enhancements to Tcl in the form of keyword extensions that facilitate the development of EEM policies. The main categories of keywords identify the detected event, the subsequent action, utility information, counter values, and system information. For more details about writing EEM policies using Tcl, see the "Writing Embedded Event Manager Policies Using Tcl" module.

Embedded Event Manager Built-In Environment Variables Used in EEM Applets

EEM built-in environment variables are a subset of the Cisco-defined environment variables and the built-in variables are available to EEM applets only. The built-in variables can be read-only or can be read and write and these variables may apply to one specific event detector or to all event detectors. The table below lists the Cisco built-in environment variables that are read-only alphabetically by event detector and subevent.

Table 5: EEM Built-In Environment Variables (Read Only)

Environment Variable	Description
All Events	
_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	An ASCII string identifier of the event type that triggered the event.
_event_pub_sec _event_pub_msec	The time, in seconds and milliseconds, at which the event was published to the EEM.
_event_severity	The severity of the event.

Environment Variable	Description
Application-Specific Event Detector	
_application_component_id	The event application component identifier.
_application_data1	The value of an environment variable, character text, or a combination of the two to be passed to an application-specific event when the event is published.
_application_data2	The value of an environment variable, character text, or a combination of the two to be passed to an application-specific event when the event is published.
_application_data3	The value of an environment variable, character text, or a combination of the two to be passed to an application-specific event when the event is published.
_application_data4	The value of an environment variable, character text, or a combination of the two to be passed to an application-specific event when the event is published.
_application_sub_system	The event application subsystem number.
_application_type	The type of application.
CLI Event Detector	
_cli_msg	The fully expanded message that triggered the CLI event.
_cli_msg_count	The number of times that a message match occurred before the event was published.
Counter Event Detector	
_counter_name	The name of the counter.
_counter_value	The value of the counter.
Enhanced Object Tracking Event Detector	
_track_number	The number of the tracked object.
_track_state	The state of the tracked object; down or up.
GOLD Event Detector	
_action_notify	The action notify information in a GOLD event flag; either false or true.

Environment Variable	Description
_event_severity	The event severity which can be one of the following; normal, minor, or major.
_gold_bl	The boot diagnostic level, which can be one of the following values: • 0: complete diagnostic • 1: minimal diagnostic • 2: bypass diagnostic
_gold_card	The card on which a GOLD failure event was detected.
_gold_cf *testnum*	Consecutive failure, where *testnum* is the test number. For example, **_gold_cf3** is the EEM built-in environment variable for consecutive failure of test 3.
_gold_ci	Card index.
_gold_cn	Card name.
_gold_ec *testnum*	Test error code, where *testnum* is the test number. For example, **_gold_ec3** is the EEM built-in environment variable for the error code of test 3.
_gold_lf *testnum*	Last fail time, where *testnum* is the test number. For example, **_gold_lf3** is the EEM built-in variable for the last fail time of test 3. The time-stamp format is *mmm dd yyyy hh*:*mm*:*ss*. For example, Mar 11 2005 08:47:00.
_gold_new_failure	The new test failure information in a GOLD event flag; either true or false.
_gold_overall_result	The overall diagnostic result, which can be one of the following values: • 0: OK • 3: minor error • 4: major error • 14: unknown result
_gold_pc	Port counts.

Environment Variable	Description
_gold_rc *testnum*	Test total run count, where *testnum* is the test number. For example, **_gold_rc3** is the EEM built-in variable for the total run count of test 3.
_gold_sn	Card serial number.
_gold_sub_card	The subcard on which a GOLD failure event was detected.
_gold_ta *testnum*	Test attribute, where *testnum* is the test number. For example, **_gold_ta3** is the EEM built-in variable for the test attribute of test 3.
_gold_tc	Test counts.
_gold_tf *testnum*	Total failure count, where *testnum* is the test number. For example, **_gold_tf3** is the EEM built-in variable for the total failure count of test 3.
_gold_tn *testnum*	Test name, where *testnum* is the test number. For example, **_gold_tn3** is the EEM built-in variable for the name of test 3.
_gold_tr *testnum*	Test result, where *testnum* is the test number. For example, **_gold_tr6** is the EEM built-in variable for test 6, where test 6 is not a per-port test and not a per-device test. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown
_gold_tr *testnum* **d** *devnum*	Per-device test result, where *testnum* is the test number and *devnum* is the device number. For example, **_gold_tr3d20** is the EEM built-in variable for the test result for test 3, device 20. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown

Environment Variable	Description
_gold_tr *testnum* **p** *portnum*	Per-port test result, where *testnum* is the test number and *portnum* is the port number. For example, **_gold_tr5p20** is the EEM built-in variable for the test result for test 5, port 20. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown
_gold_tt	The testing type, which can be one of the following: • 1: a boot diagnostic • 2: an on-demand diagnostic • 3: a schedule diagnostic • 4: a monitoring diagnostic
Interface Counter Event Detector	
_interface_is_increment	A value to indicate whether the current interface counter value is an absolute value (0) or an increment value (1).
_interface_name	The name of the interface to be monitored.
_interface_parameter	The name of the interface counter to be monitored.
_interface_value	A value with which the current interface counter value is compared.
None Event Detector	
_event_id	A value of 1 indicates an insertion event; a value of 2 indicates a removal event.

Environment Variable	Description
_none_argc **_none_arg1** **_none_arg2** **_none_arg3** **_none_arg4** **_none_arg5** **_none_arg6** **_none_arg7** **_none_arg8** **_none_arg9** **_none_arg10** **_none_arg11** **_none_arg12** **_none_arg13** **_none_arg14** **_none_arg15**	The parameters that are passed from the XML SOAP command to the script.
OIR Event Detector	
_oir_event	A value of 1 indicates an insertion event; a value of 2 indicates a removal event.
_oir_slot	The slot number for the OIR event.
Resource Event Detector	
_resource_configured_threshold	The configured ERM threshold.
_resource_current_value	The current value reported by ERM.
_resource_dampen_time	The ERM dampen time, in nanoseconds.
_resource_direction	The ERM event direction. The event direction can be one of the following: up, down, or no change.
_resource_level	The ERM event level. The four event levels are normal, minor, major, and critical.
_resource_notify_data_flag	The ERM notify data flag.
_resource_owner_id	The ERM resource owner ID.

Environment Variable	Description
_resource_policy_id	The ERM policy ID.
_resource_policy_violation_flag	The ERM policy violation flag; either false or true.
_resource_time_sent	The ERM event time, in nanoseconds.
_resource_user_id	The ERM resource user ID.
RF Event Detector	
_rf_event	A value of 0 indicates that this is not an RF event; a value of 1 indicates an RF event.
RPC Event Detector	
_rpc_event	A value of 0 indicates that there is no error; a value of 1 to 83 indicates error.
_rpc_argc **_rpc_arg0** **_rpc_arg1** **_rpc_arg2** **_rpc_arg3** **_rpc_arg4** **_rpc_arg5** **_rpc_arg6** **_rpc_arg7** **_rpc_arg8** **_rpc_arg9** **_rpc_arg10** **_rpc_arg11** **_rpc_arg12** **_rpc_arg13** **_rpc_arg14**	The parameters that are passed from the XML SOAP command to the applet.
SNMP Event Detector	
_snmp_exit_event	A value of 0 indicates that this is not an exit event; a value of 1 indicates an exit event.
_snmp_oid	The SNMP object ID that caused the event to be published.

Environment Variable	Description
_snmp_oid_delta_val	The actual incremental difference between the value of the current SNMP object ID and the value when the event was last triggered.
_snmp_oid_val	The SNMP object ID value when the event was published.
SNMP Notification Event Detector	
_snmp_notif_oid	A user specified object ID.
_snmp_notif_oid_val	A user specified object ID value.
_snmp_notif_src_ip_addr	The source IP address of the SNMP Protocol Data Unit (PDU).
_snmp_notif_dest_ip_addr	The destination IP address of the SNMP PDU.
_x_x_x_x_x_x_x(varbinds)	The SNMP PDU varbind information.
_snmp_notif_trunc_vb_buf	Indicates whether the varbind information has been truncated due to the lack of space in the buffer.
Syslog Event Detector	
_syslog_msg	The syslog message that caused the event to be published.
System Manager (Process) Event Detector	
_process_dump_count	The number of times that a Posix process was dumped.
_process_exit_status	The status of the Posix process at exit.
_process_fail_count	The number of times that a Posix process failed.
_process_instance	The instance number of the Posix process.
_process_last_respawn	The Posix process that was last respawned.
_process_node_name	The node name of the Posix process.
_process_path	The path of the Posix process.
_process_process_name	The name of the Posix process.
_process_respawn_count	The number of times that a Posix process was respawned.

Environment Variable	Description
Timer Event Detector	
_timer_remain	The time available before the timer expires. **Note** This environment variable is not available for the CRON timer.
_timer_time	The time at which the last event was triggered.
_timer_type	The type of timer.
Watchdog System Monitor (IOSWDSysMon) Event Detector	
_ioswd_node	The slot number for the Route Processor (RP) reporting node.
_ioswd_num_subs	The number of subevents present.
All Watchdog System Monitor (IOSWDSysMon) Subevents	
_ioswd_sub1_present _ioswd_sub2_present	A value to indicate whether subevent 1 or subevent 2 is present. A value of 1 means that the subevent is present; a value of 0 means that the subevent is not present.
_ioswd_sub1_type _ioswd_sub2_type	The event type, either cpu_proc or mem_proc.
Watchdog System Monitor (IOSWDSysMon) cpu_proc Subevents	
_ioswd_sub1_path _ioswd_sub2_path	A process name of subevents.
_ioswd_sub1_period _ioswd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_ioswd_sub1_pid _ioswd_sub2_pid	The process identifier of subevents.
_ioswd_sub1_taskname _ioswd_sub2_taskname	The task name of subevents.
_ioswd_sub1_value _ioswd_sub2_value	The CPU utilization of subevents measured as a percentage.
Watchdog System Monitor (IOSWDSysMon) mem_proc Subevents	

Environment Variable	Description
_ioswd_sub1_diff _ioswd_sub2_diff	A percentage value of the difference that triggered the event. **Note** This variable is set only when the **_ioswd_sub1_is_percent** or **_ioswd_sub2_is_percent** variable contains a value of 1.
_ioswd_sub1_is_percent _ioswd_sub2_is_percent	A number that identifies whether the value is a percentage. A value of 0 means that the value is not a percentage; a value of 1 means that the value is a percentage.
_ioswd_sub1_path _ioswd_sub2_path	The process name of subevents.
_ioswd_sub1_pid _ioswd_sub2_pid	The process identifier of subevents.
_ioswd_sub1_taskname _ioswd_sub2_taskname	The task name of subevents.
_ioswd_sub1_value _ioswd_sub2_value	The CPU utilization of subevents measured as a percentage.
Watchdog System Monitor (WDSysMon) Event Detector	
_wd_sub1_present _wd_sub2_present	A value to indicate whether subevent 1 or subevent 2 is present. A value of 1 means that the subevent is present; a value of 0 means that the subevent is not present.
_wd_num_subs	The number of subevents present.
_wd_sub1_type _wd_sub2_type	The event type: cpu_proc, cpu_tot, deadlock, dispatch_mgr, mem_proc, mem_tot_avail, or mem_tot_used.
Watchdog System Monitor (WDSysMon) cpu_proc Subevents	
_wd_sub1_node _wd_sub2_node	The slot number for the subevent RP reporting node.
_wd_sub1_period _wd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_wd_sub1_procname _wd_sub2_procname	The process name of subevents.
_wd_sub1_value _wd_sub2_value	The CPU utilization of subevents measured as a percentage.

Environment Variable	Description
Watchdog System Monitor (WDSysMon) cpu_tot Subevents	
_wd_sub1_node _wd_sub2_node	The slot number for the subevent RP reporting node.
_wd_sub1_period _wd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_wd_sub1_value _wd_sub2_value	The CPU utilization of subevents measured as a percentage.
Watchdog System Monitor (WDSysMon) deadlock Subevents	
_wd_sub1_entry_ [1-N]**_b_node** **_wd_sub2_entry_**[1-N]**_b_node**	The slot number for the subevent RP reporting node.
_wd_sub1_entry_ [1-N]**_b_pid** **_wd_sub2_entry_**[1-N]**_b_pid**	The process identifier of subevents.
_wd_sub1_entry_ [1-N]**_b_procname** **_wd_sub2_entry_**[1-N]**_b_procname**	The process name of subevents.
_wd_sub1_entry_ [1-N]**_b_tid** **_wd_sub2_entry_**[1-N]**_b_tid**	The time identifier of subevents.
_wd_sub1_entry_ [1-N]**_node** **_wd_sub2_entry_**[1-N]**_node**	The slot number for the subevent RP reporting node.
_wd_sub1_entry_ [1-N]**_pid** **_wd_sub2_entry_**[1-N]**_pid**	The process identifier of subevents.
_wd_sub1_entry_ [1-N]**_procname** **_wd_sub2_entry_**[1-N]**_procname**	The process name of subevents.
_wd_sub1_entry_ [1-N]**_state** **_wd_sub2_entry_**[1-N]**_state**	The time identifier of subevents.
_wd_sub1_entry_ [1-N]**_tid** **_wd_sub2_entry_**[1-N]**_tid**	The time identifier of subevents.
_wd_sub1_num_entries _wd_sub2_num_entries	The number of subevents.
Watchdog System Monitor (WDSysMon) dispatch manager Subevents	
_wd_sub1_node _wd_sub2_node	The slot number for the subevent RP reporting node.

Environment Variable	Description
_wd_sub1_period _wd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_wd_sub1_procname _wd_sub2_procname	The process name of subevents.
_wd_sub1_value _wd_sub2_value	The CPU utilization of subevents measured as a percentage.
Watchdog System Monitor (WDSysMon) mem_proc Subevents	
_wd_sub1_diff _wd_sub2_diff	A percentage value of the difference that triggered the event. **Note** This variable is set only when the **_wd_sub1_is_percent** or **_wd_sub2_is_percent** variable contains a value of 1.
_wd_sub1_is_percent _wd_sub2_is_percent	A number that identifies whether the value is a percentage. A value of 0 means that the value is not a percentage; a value of 1 means that the value is a percentage.
_wd_sub1_node _wd_sub2_node	The slot number for the subevent RP reporting node.
_wd_sub1_period _wd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_wd_sub1_pid _wd_sub2_pid	The process identifier of subevents.
_wd_sub1_procname _wd_sub2_procname	The process name of subevents.
_wd_sub1_value _wd_sub2_value	The CPU utilization of subevents measured as a percentage.
Watchdog System Monitor (WDSysMon) mem_tot_avail and mem_tot_used Subevents	
_wd_sub1_avail _wd_sub2_avail	The memory available for subevents.
_wd_sub1_diff _wd_sub2_diff	A percentage value of the difference that triggered the event. **Note** This variable is set only when the **_wd_sub1_is_percent** or **_wd_sub2_is_percent** variable contains a value of 1.

Environment Variable	Description
_wd_sub1_is_percent _wd_sub2_is_percent	A number that identifies whether the value is a percentage. A value of 0 means that the value is not a percentage; a value of 1 means that the value is a percentage.
_wd_sub1_node _wd_sub2_node	The slot number for the subevent RP reporting node.
_wd_sub1_period _wd_sub2_period	The time period, in seconds and optional milliseconds, used for measurement in subevents.
_wd_sub1_value _wd_sub2_value	The CPU utilization of subevents measured as a percentage.
_wd_sub1_used _wd_sub2_used	The memory used by subevents.

How to Write EEM Policies Using the Cisco IOS CLI

Registering and Defining an Embedded Event Manager Applet

Perform this task to register an applet with Embedded Event Manager and to define the EEM applet using the Cisco IOS CLI **event** and **action** commands. Only one **event** command is allowed in an EEM applet. Multiple **action** commands are permitted. If no **event** and no **action** commands are specified, the applet is removed when you exit configuration mode.

The SNMP event detector and the syslog **action** commands used in this task are just representing any event detector and **action** commands. For examples using other event detectors and **action** commands, see the Embedded Event Manager Applet Configuration Examples, on page 88.

EEM Environment Variables

EEM environment variables for EEM policies are defined using the EEM **event manager environment** configuration command. By convention, all Cisco EEM environment variables begin with " _ ". In order to avoid future conflict, customers are urged not to define new variables that start with " _ ".

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

For example, you can create EEM policies that can send e-mails when an event occurs. The table below describes the e-mail-specific environment variables that can be used in EEM policies.

Table 6: EEM E-mail-Specific Environmental Variables

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

Alphabetical Order of EEM Action Labels

An EEM action label is a unique identifier that can be any string value. Actions are sorted and run in ascending alphanumeric (lexicographical) key sequence using the label as the sort key. If you are using numbers as labels be aware that alphanumerical sorting will sort 10.0 after 1.0, but before 2.0, and in this situation we recommend that you use numbers such as 01.0, 02.0, and so on, or use an initial letter followed by numbers.

SUMMARY STEPS

1. **enable**
2. **show event manager environment** [**all**| *variable-name*]
3. **configure terminal**
4. **event manager environment** *variable-name string*
5. Repeat Alphabetical Order of EEM Action Labels for all the required environment variables.
6. **event manager applet** *applet-name*
7. Do one of the following:
   - **event snmp oid** *oid-value* **get-type** {**exact**| **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb**| **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value*
8. **action** *label* **cli command** *cli-string* [**pattern** *pattern-string*]
9. **action** *label* **syslog** [**priority** *priority-level*] **msg** *msg-text* **facility** *string*
10. **action** *label* **mail server** *server-address* **to** *to-address* **from** *from-address* [**cc** *cc-address*] **subject** *subject* **body** *body-text*
11. Add more action commands as required.
12. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager environment** [**all**| *variable-name*] **Example:** `Device# show event manager environment all`	(Optional) Displays the name and value of EEM environment variables. • The optional **all** keyword displays all the EEM environment variables. • The optional *variable-name*argument displays information about the specified environment variable.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**event manager environment** *variable-name string*	Configures the value of the specified EEM environment variable.

	Command or Action	Purpose
	Example: `Device(config)# event manager environment _email_to engineering@example.com`	• In this example, the environment variable that holds the e-mail address to which e-mail is sent is set to engineering@example.com.
Step 5	Repeat Alphabetical Order of EEM Action Labels for all the required environment variables.	Repeat Alphabetical Order of EEM Action Labels to configure all the environment variables required by the policy to be registered in Alphabetical Order of EEM Action Labels.
Step 6	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet memory-fail`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 7	Do one of the following: • **event snmp oid** *oid-value* **get-type** {**exact**| **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb**| **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value* **Example:** `Device(config-applet)# event snmp oid 1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact entry-op lt entry-val 5120000 poll-interval 90`	Specifies the event criteria that cause the EEM applet to run. • In this example, an EEM event is triggered when free memory falls below the value of 5120000. • Exit criteria are optional, and if not specified, event monitoring is reenabled immediately.

Step 8	**action** *label* **cli command** *cli-string* [**pattern** *pattern-string*]
	Example:
	`Device(config-applet)# action 1.0 cli command "enable"`
	Example:
	`Device(config-applet)# action 2.0 cli command "clear counters Ethernet0/1" pattern "confirm"`
	Example:
	`Device(config-applet)# action 3.0 cli command "y"`
		Specifies the action of executing a Cisco IOS CLI command when an EEM applet is triggered.
		The **pattern** keyword is optional and is used only when the command string solicits input. The **action cli**command ends when the solicited prompt as specified in the optional **pattern** keyword is received. You are required to specify a regular expression pattern that will match the next solicited prompt. Specification of an incorrect pattern will cause the **action cli** command to wait forever until the applet execution times out due to the maxrun timer expiration.
		• The action taken is to specify an EEM applet to run when the **pattern** keyword specifies the *confirm* argument for the **clear counters Ethernet0/1** command. In this case the command string solicits input, such as “confirm,” which has to be completed with a “yes” or a “no” input.

	Command or Action	Purpose

Step 9	**action** *label* **syslog** [**priority** *priority-level*] **msg** *msg-text* **facility** *string*
	
	Example:
	
	`Device(config-applet)# action 1.0 syslog priority critical msg "Memory exhausted; current available memory is $_snmp_oid_val bytes"`
	
	Example:
	
	`Device(config-applet)# action 1.0 syslog priority errors facility EEM-FAC message “TEST MSG”`
		Specifies the action to be taken when an EEM applet is triggered.
		
		In this example, the action taken is to write a message to syslog.
		
		• The optional **priority** keyword specifies the priority level of the syslog messages. If selected, the *priority-level* argument must be defined.
		
		• The *msg-text* argument can be character text, an environment variable, or a combination of the two.
		
		• The **facility** keyword specifies the location of generated message
		
		• The *string* argument can be character text, an environment variable, or a combination of the two.

Step 10	**action** *label* **mail server** *server-address* **to** *to-address* **from** *from-address* [**cc** *cc-address*] **subject** *subject* **body** *body-text* **Example:** `Device(config-applet)# action 2.0 mail server 192.168.1.10 to engineering@example.com from devtest@example.com subject "Memory failure" body "Memory exhausted; current available memory is $_snmp_oid_val bytes"`	Specifies the action of sending a short e-mail when an EEM applet is triggered. • The *server-address* argument specifies the fully qualified domain name of the e-mail server to be used to forward the e-mail. • The *to-address* argument specifies the e-mail address where the e-mail is to be sent. • The *from-address* argument specifies the e-mail address from which the e-mail is sent. • The *subject* argument specifies the subject line content of the e-mail as an alphanumeric string. • The *body-text* argument specifies the text content of the e-mail as an alphanumeric string.
Step 11	Add more action commands as required.	--
Step 12	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.

Troubleshooting Tips

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

Registering and Defining an Embedded Event Manager Policy to Run Manually

There are two ways to manually run an EEM policy. EEM usually schedules and runs policies on the basis of an event specification that is contained within the policy itself. The **event none** command allows EEM to identify an EEM policy that can be manually triggered. To run the policy, use either the **action policy** command in applet configuration mode or the **event manager run** command in privileged EXEC mode.

Perform this task to register an EEM policy to be run manually using the **event manager run** command. For an example of how to manually run a policy using the **action policy** command, see the Embedded Event Manager Manual Policy Execution Examples, on page 93.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event none**
5. **action** *label* **syslog** [**priority** *priority-level*] **msg** *msg-text* **facility** *string*
6. **end**
7. **event manager run** *applet-name*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet manual-policy`	Registers the applet with the Embedded Event Manager and enters applet configuration mode.
Step 4	**event none** **Example:** `Device(config-applet)# event none`	Specifies that an EEM policy is to be registered with the EEM and can be run manually.

	Command or Action	Purpose
Step 5	**action** *label* **syslog** [**priority** *priority-level*] **msg** *msg-text* **facility** *string* **Example:** `Device(config-applet)# action 1.0 syslog msg "Manual-policy triggered"`	Specifies the action to be taken when an EEM applet is triggered. In this example, the action to be taken is to write a message to syslog. • The optional **priority** keyword specifies the priority level of the syslog messages. If selected, the *priority-level* argument must be defined. • The *msg-text* argument can be character text, an environment variable, or a combination of the two. • The **facility** keyword specifies the location of generated message. • The *string* argument can be character text, an environment variable, or a combination of the two.
Step 6	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.
Step 7	**event manager run** *applet-name* **Example:** `Device# event manager run manual-policy`	Manually runs a registered EEM policy.

Unregistering Embedded Event Manager Policies

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

SUMMARY STEPS

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

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy registered** [**description** [*policy-name*] | **detailed** *policy-filename* [**system** | **user**] | [**event-type** *event-name*] [**system** | **user**] [**time-ordered** | **name-ordered**]] **Example:** `Device# show event manager policy registered`	(Optional) Displays the EEM policies that are currently registered. • The optional **system**and **user** keywords display the registered system and user policies. • If no keywords are specified, EEM registered policies for all event types are displayed in time order.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**no event manager policy** *policy-filename* **Example:** `Device(config)# no event manager policy IPSLAping1`	Removes the EEM policy from the configuration, causing the policy to be unregistered.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 6	Repeat Step 2 to ensure that the policy has been removed. **Example:** `Device# show event manager policy registered`	--

Examples

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

```
Device# show event manager policy registered
No.  Class   Type    Event Type          Trap  Time Registered           Name
1    applet  system  snmp                Off   Fri Aug 12 17:42:52 2005  IPSLAping1
```

```
 oid {1.3.6.1.4.1.9.9.42.1.2.9.1.6.4} get-type exact entry-op eq entry-val {1}
 exit-op eq exit-val {2} poll-interval 90.000
 action 1.0 syslog priority critical msg "Server IPecho Failed: OID=$_snmp_oid_val"
 action 1.1 snmp-trap strdata "EEM detected server reachability failure to 10.1.88.9"
 action 1.2 publish-event sub-system 88000101 type 1 arg1 "10.1.88.9" arg2 "IPSLAEcho"
arg3 "fail"
 action 1.3 counter name _IPSLA1F op inc value 1
2    applet  system  snmp                Off   Thu Sep 15 05:57:16 2005  memory-fail
 oid {1.3.6.1.4.1.9.9.48.1.1.1.6.1} get-type exact entry-op lt entry-val {5120000}
poll-interval 90
 action 1.0 syslog priority critical msg Memory exhausted; current available memory is
$_snmp_oid_val bytes
 action 2.0 force-switchover
```

In the following example, the **show event manager policy registered** privileged EXEC command is used to show that applet IPSLAping1 has been removed after entering the **no event manager policy** command:

```
Device# show event manager policy registered
No.  Class   Type    Event Type          Trap  Time Registered           Name
1    applet  system  snmp                Off   Thu Sep 15 05:57:16 2005  memory-fail
 oid {1.3.6.1.4.1.9.9.48.1.1.1.6.1} get-type exact entry-op lt entry-val {5120000}
poll-interval 90
 action 1.0 syslog priority critical msg Memory exhausted; current available memory is
$_snmp_oid_val bytes
 action 2.0 force-switchover
```

Suspending All Embedded Event Manager Policy Execution

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

SUMMARY STEPS

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

DETAILED STEPS

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

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

Configuring and Tracking a Stub Object Using Embedded Event Manager

Perform this task to create a stub object, set the state of the stub object, and configure an EEM applet to be run when the tracked object changes. Actions are specified within the EEM applet to both set and read the state of the object. This task allows EEM to define an enhanced object tracking (EOT) object that may be manipulated by other EOT clients. An EEM policy can be a trigger for any EOT object including objects defined for other EOT clients or for an object defined by EEM.

Enhanced Object Tracking

Object tracking was first introduced into the Hot Standby Router Protocol (HSRP) as a simple tracking mechanism that allowed you to track the interface line-protocol state only. Enhanced object tracking provides complete separation between the objects to be tracked and the action to be taken by a client when a tracked object changes. Thus, several clients such as EEM, VRRP, or GLBP can register their interest with the tracking process, track the same object, and each take different action when the object changes.

Each tracked object is identified by a unique number that is specified on the tracking command-line interface (CLI). Client processes use this number to track a specific object. The tracking process periodically polls the tracked objects and notes any change of value. The changes in the tracked object are communicated to interested client processes, either immediately or after a specified delay. The object values are reported as either up or down.

The EOT event detector publishes an event when the tracked object changes.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **track** *object-number* **stub-object**
4. **default-state** {**up**| **down**}
5. **exit**
6. **event manager applet** *applet-name*
7. **event** [*label*] **track** *object-number*[**state**{**up**| **down**| **any**}]
8. **action** *label* **track set** *object-number* **state** {**up**| **down**}
9. **action** *label* **track read** *object-number*
10. **end**
11. **show track** [*object-number*[**brief**]]

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**track** *object-number* **stub-object** **Example:** `Device(config)# track 2 stub-object`	Creates a stub object to be tracked using EEM and enters tracking configuration mode. • Use the *object-number* argument to assign a number to the tracked object.
Step 4	**default-state** {**up**| **down**} **Example:** `Device(config-track)# default-state up`	Sets the default state for a stub object. • In this example, the default state of the object is set to up.
Step 5	**exit** **Example:** `Device(config-track)# exit`	Exits tracking configuration mode and returns to global configuration mode.

	Command or Action	Purpose
Step 6	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet track-two`	Registers an applet with EEM and enters applet configuration mode.
Step 7	**event** [*label*] **track** *object-number*[**state**{**up**| **down**| **any**}] **Example:** `Device(config-applet)# event track 2 state down`	Specifies the event criteria that cause the EEM applet to run. • In this example, an EEM event is triggered when the Cisco IOS Object Tracking subsystem reports that tracked object number 2 transitions from an up state to a down state.
Step 8	**action** *label* **track set** *object-number* **state** {**up**| **down**} **Example:** `Device(config-applet)# action 1.0 track set 2 state up`	Specifies the action to be taken when an EEM applet is triggered. • In this example, the action to be taken is to set the state of tracked object number 2 to up.
Step 9	**action** *label* **track read** *object-number* **Example:** `Device(config-applet)# action 2.0 track read 2`	Specifies the action to be taken when an EEM applet is triggered. • In this example, the action to be taken is to read the state of tracked object number 2. • The _track_state read-only variable gets set when this command is run.
Step 10	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.
Step 11	**show track** [*object-number*[**brief**]] **Example:** `Device# show track 2`	(Optional) Displays information about objects that are tracked by the tracking process. • The optional *object-number* argument displays tracking information for a specified object. • The optional **brief** keyword displays a single line of information.

Examples

In the following example, the **show track** privileged EXEC command is used to display information about tracked object number 2.

```
Device# show track 2
Track 2
  Stub-object
  State is Up
    1 change, last change 00:00:04, by Undefined
```

Displaying Embedded Event Manager History Data

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

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **configure terminal**
Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 3 **event manager history size** {**events** | **traps**} [*size*]
Use this command to change the size of the EEM event history table or the size of the EEM SNMP trap history table. In the following example, the size of the EEM event history table is changed to 30 entries:

Example:

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

Step 4 **exit**
Exits global configuration mode and returns to privileged EXEC mode.

Example:

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

Step 5 **show event manager history events** [**detailed**] [**maximum** *number*]
Use this command to display detailed information about each EEM event, for example:

Example:

```
Device# show event manager history events
No.  Time of Event             Event Type          Name
1    Fri Aug13  21:42:57 2004  snmp                applet: SAAping1
2    Fri Aug13  22:20:29 2004  snmp                applet: SAAping1
3    Wed Aug18  21:54:48 2004  snmp                applet: SAAping1
4    Wed Aug18  22:06:38 2004  snmp                applet: SAAping1
5    Wed Aug18  22:30:58 2004  snmp                applet: SAAping1
6    Wed Aug18  22:34:58 2004  snmp                applet: SAAping1
7    Wed Aug18  22:51:18 2004  snmp                applet: SAAping1
8    Wed Aug18  22:51:18 2004  application         applet: CustApp1
```

Step 6 **show event manager history traps** {**server** | **policy**}
Use this command to display the EEM SNMP traps that have been sent either from the EEM server or from an EEM policy. In the following example, the EEM SNMP traps that were triggered from within an EEM policy are displayed.

Example:

```
Device# show event manager history traps policy
No.  Time                       Trap Type           Name
1    Wed Aug18  22:30:58 2004  policy              EEM Policy Director
2    Wed Aug18  22:34:58 2004  policy              EEM Policy Director
3    Wed Aug18  22:51:18 2004  policy              EEM Policy Director
```

Displaying Embedded Event Manager Registered Policies

Perform this optional task to display registered EEM policies.

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

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

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

Example:

```
Device# show event manager policy registered time-ordered
No.  Type    Event Type          Time                    Registered Name
1    applet  snmp                Thu May30 05:57:16 2004 memory-fail
 oid {1.3.6.1.4.1.9.9.48.1.1.1.6.1} get-type exact entry-op lt entry-val
{5120000} poll-interval 90
action 1.0 syslog priority critical msg "Memory exhausted; current available memory
is $_snmp_oid_val bytes"
 action 2.0 force-switchover
2    applet  syslog              Wed Jul16 00:05:17 2004 intf-down
 pattern {.*UPDOWN.*Ethernet1/0.*}
 action 1.0 cns-event msg "Interface state change: $_syslog_msg"
```

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

Example:

```
Device# show event manager policy registered name-ordered
No.  Type    Event Type          Time Registered          Name
1    applet  syslog              Wed Jul16  00:05:17 2004 intf-down
 pattern {.*UPDOWN.*Ethernet1/0.*}
 action 1.0 cns-event msg "Interface state change: $_syslog_msg"
2    applet  snmp                Thu May30 05:57:16 2004   memory-fail
 oid {1.3.6.1.4.1.9.9.48.1.1.1.6.1} get-type exact entry-op lt entry-val
{5120000} poll-interval 90
 action 1.0 syslog priority critical msg "Memory exhausted; current available memory
is $_snmp_oid_val bytes"
 action 2.0 force-switchover
```

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

Example:

```
Device# show event manager policy registered event-type syslog
No.  Type    Event Type          Time Registered           Name
1    applet  syslog              Wed Jul16  00:05:17 2004 intf-down
 pattern {.*UPDOWN.*Ethernet1/0.*}
 action 1.0 cns-event msg "Interface state change: $_syslog_msg"
```

Configuring Event SNMP Notification

Perform this task to configure SNMP notifications.

Before You Begin

- SNMP event manager must be configured using the **snmp-server manager** command.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event** [**tag** *event-tag*] **snmp-notification oid** *oid-string* **oid-val** *comparison-value* **op** *operator* [**maxrun** *maxruntime-number*] [**src-ip-address** *ip-address*] [**dest-ip-address** *ip-address*] [**default** *seconds*] [**direction** {**incoming** | **outgoing**}] [**msg-op** {**drop** | **send**}]
5. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet snmp`	Registers the applet with the event manager server and enters applet configuration mode.
Step 4	**event** [**tag** *event-tag*] **snmp-notification oid** *oid-string* **oid-val** *comparison-value* **op** *operator* [**maxrun** *maxruntime-number*] [**src-ip-address** *ip-address*] [**dest-ip-address** *ip-address*] [**default** *seconds*] [**direction** {**incoming**	**outgoing**}] [**msg-op** {**drop**	**send**}] **Example:** `Device(config-applet)# event snmp-notification dest-ip-address 192.168.1.1 oid 1 op eq oid-val 10`	Specifies the event criteria for an Embedded Event Manager (EEM) applet that is run by sampling Simple Network Management Protocol (SNMP) notification.

	Command or Action	Purpose
Step 5	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.

Configuring Multiple Event Support

The multiple event support feature adds the ability to register multiple events in the EEM server. The multiple event support involves one or more event occurrences, one or more tracked object states, and a time period for the event to occur. The event parameters are specified in the CLI commands. The data structure to handle multiple events contains multiple event identifiers and correlation logic. This data is used to register multiple events in the EEM Server.

Setting the Event Configuration Parameters

The **trigger** command enters the trigger applet configuration mode and specifies the multiple event configuration statements for EEM applets. The trigger statement is used to relate multiple event statement using the *tag* argument specified in each event statement. The events are raised based on the specified parameters.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event** [**tag** *event-tag*] **cli pattern** *regular-expression* **sync** {**yes** | **no skip** {**yes** | **no**}} [**occurs** *num-occurrences*] [**period** *period-value*] [**maxrun** *maxruntime-number*]
5. **trigger** [**occurs** *occurs-value*] [**period** *period-value*] [**period-start** *period-start-value*] [**delay** *delay-value*]
6. **correlate** {**event** *event-tag* | **track** *object-number*} [*boolean-operator* **event** *event-tag*]
7. **attribute tag** *event-tag* [**occurs** *occurs-value*]
8. **action** *label* **cli command** *cli-string*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet EventInterface`	Registers an applet with EEM and enters applet configuration mode.
Step 4	**event** [**tag** *event-tag*] **cli pattern** *regular-expression* **sync** {**yes** | **no skip** {**yes** | **no**}} [**occurs** *num-occurrences*] [**period** *period-value*] [**maxrun** *maxruntime-number*] **Example:** `Device(config-applet)# event tag 1.0 cli pattern "show bgp all" sync yes occurs 32 period 60 maxrun 60`	Specifies the event criteria for an EEM applet that is run by matching a Cisco IOS command-line interface (CLI) command.
Step 5	**trigger** [**occurs** *occurs-value*] [**period** *period-value*] [**period-start** *period-start-value*] [**delay** *delay-value*] **Example:** `Device(config-applet)# trigger occurs 1 period-start "0 8 * * 1-5" period 60`	Specifies the complex event configuration parameters for an EEM applet.
Step 6	**correlate** {**event** *event-tag* | **track** *object-number*} [*boolean-operator* **event** *event-tag*] **Example:** `Device(config-applet)# correlate event 1.0 or event 2.0`	Specifies a complex event correlation in the trigger mode for an EEM applet. **Note** When "and" is used to group events such as traps or syslog messages, then the default trigger occurrence window is three minutes.
Step 7	**attribute tag** *event-tag* [**occurs** *occurs-value*] **Example:** `Device(config-applet)# attribute tag 1.0 occurs 1`	Specifies up to eight attribute statements to build a complex event for an EEM applet.
Step 8	**action** *label* **cli command** *cli-string* **Example:** `Device(config-applet)# action 1.0 cli command "show pattern"`	Specifies the action of executing a CLI command when an EEM applet is triggered.

Examples

In the following example, applet is run if the **show bgp all** CLI command and any syslog message that contains the string "COUNT" occurred within a period 60 seconds.

```
event manager applet delay_50
 event tag 1.0 cli pattern "show bgp all" sync yes occurs 32 period 60 maxrun 60
 event tag 2.0 syslog pattern "COUNT"
 trigger occurs 1 delay 50
  correlate event 1.0 or event 2.0
  attribute tag 1.0 occurs 1
  attribute tag 2.0 occurs 1
 action 1.0 cli command "show memory"
 action 2.0 cli command "enable"
 action 3.0 cli command "config terminal"
 action 4.0 cli command " ip route 192.0.2.0 255.255.255.224 192.0.2.12"
 action 91.0 cli command "exit"
 action 99.0 cli command "show ip route | incl 192.0.2.5"
```

Configuring EEM Class-Based Scheduling

To schedule Embedded Event Manager (EEM) policies and set policy scheduling options, perform this task. In this task, two EEM execution threads are created to run applets assigned to the default class.

The EEM policies will be assigned a class using the **class** keyword when they are registered. EEM policies registered without a class will be assigned to the default class. Threads that have default class, will service the default class when the thread is available for work. Threads that are assigned specific class letters will service any policy with a matching class letter when the thread is available for work.

If there is no EEM execution thread available to run the policy in the specified class and a scheduler rule for the class is configured, the policy will wait until a thread of that class is available for execution. Synchronous policies that are triggered from the same input event should be scheduled in the same execution thread.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager scheduler** {**applet** | **axp** | **call-home**} **thread class** *class-options* **number** *thread-number*
4. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

	Command or Action	Purpose
Step 3	**event manager scheduler** {**applet** | **axp** | **call-home**} **thread class** *class-options* **number** *thread-number* **Example:** `Device(config)# event manager scheduler applet thread class default number 2`	Schedules EEM policies and sets policy scheduling options. • In this example, two EEM execution threads are created to run applets assigned to the default class.
Step 4	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Holding a Scheduled EEM Policy Event or Event Queue

To hold a scheduled EEM policy event or event queue in the EEM scheduler, perform this task. In this task, all pending EEM policies are displayed. A policy identified using a job ID of 2 is held in the EEM scheduler, and the final step shows that the policy with a job ID of 2 has changed status from pending to held.

SUMMARY STEPS

1. **enable**
2. **show event manager policy pending** [**queue-type**{**applet** \| **call-home** \| **axp** \| **script**} **class** *class-options* \| **detailed**]
3. **event manager scheduler hold** {**all**\| **policy** *job-id* \| **queue-type** {**applet** \| **call-home** \| **axp** \| **script**} **class** *class-options*} [**processor** {**rp_primary**\| **rp_standby**}]
4. **show event manager policy pending** [**queue-type**{**applet** \| **call-home** \| **axp** \| **script**} **class** *class-options* \| **detailed**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy pending** [**queue-type**{**applet** | **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]	Displays the pending EEM policies.

	Command or Action	Purpose
	Example: `Device# show event manager policy pending`	
Step 3	**event manager scheduler hold** {**all**| **policy** *job-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**} **class** *class-options*} [**processor** {**rp_primary**| **rp_standby**}] **Example:** `Device# event manager scheduler hold policy 2`	Holds a scheduled EEM policy event or event queue in the EEM scheduler. • In this example, a policy with a job ID of 2 is put on hold.
Step 4	**show event manager policy pending** [**queue-type**{**applet** | **call-home** | **axp** | **script**} **class** *class-options* | **detailed** **Example:** `Device# show event manager policy pending`	Displays the status of EEM policy put on hold in Step 3 as held, along with other pending policies.

Examples

The following example shows how to view all pending EEM policies and to hold the EEM policy with a job ID of 2.

```
Device# show event manager policy pending
no. job id status time of event             event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2   2      pend   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
Device# event manager scheduler hold policy 2
Device# show event manager policy pending

no. job id status time of event             event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2  2      held   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
```

Resuming Execution of EEM Policy Events or Event Queues

To resume the execution of specified EEM policies, perform this task. In this task, the policy that was put on hold in the Holding a Scheduled EEM Policy Event or Event Queue task is now allowed to resume execution.

SUMMARY STEPS

1. **enable**
2. **show event manager policy pending**
3. **event manager scheduler release** {**all** | **policy** *policy-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**}} **class** *class-options* [**processor** {*rp_primary* | *rp_standby*}]
4. **show event manager policy pending**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays the pending and held EEM policies. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.
Step 3	**event manager scheduler release** {**all** | **policy** *policy-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**}} **class** *class-options* [**processor** {*rp_primary* | *rp_standby*}] **Example:** `Device# event manager scheduler release policy 2`	Resumes execution of specified EEM policies. • The example shows how to resume the execution of the policy with job ID of 2.
Step 4	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays the status of the EEM policy resumed in Step 3 as pending, along with other pending policies. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.

Examples

The following example shows how to view all pending EEM policies, to specify the policy that will resume execution, and to see that the policy is now back in a pending status.

```
Device# show event manager policy pending

no. job id status time of event             event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2  2      held   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
```

```
Rotuer# event manager scheduler release policy 2
Rotuer# show event manager policy pending
no. job id status time of event            event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2   2      pend   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
```

Clearing Pending EEM Policy Events or Event Queues

Perform this task to clear EEM policies that are executing or pending execution. In this task, the EEM policy with a job ID of 2 is cleared from the pending queue. The **show event manager policy pending** command is used to display the policies that are pending before and after the policy is cleared.

SUMMARY STEPS

1. **enable**
2. **show event manager policy pending**
3. **event manager scheduler clear** {**all** | **policy** *job-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**} **class** *class-options*} [**processor** {**rp_primary** | **rp_standby**}]
4. **show event manager policy pending**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays the pending EEM policies. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.
Step 3	**event manager scheduler clear** {**all** | **policy** *job-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**} **class** *class-options*} [**processor** {**rp_primary** | **rp_standby**}] **Example:** `Device# event manager scheduler clear policy 2`	Clears EEM policies that are executing or pending execution. • In this example, the EEM policy with a job ID of 2 is cleared from the pending queue.
Step 4	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays all the pending EEM policies except the policy cleared in Step 3. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.

	Command or Action	Purpose

Examples

The following example shows how to clear the EEM policy with a job ID of 2 that was pending execution. The **show** commands are used to display the policies that are pending before and after the policy is cleared.

```
Device# show event manager policy pending
no. job id status time of event             event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2   2      pend   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three

Device# event manager scheduler clear policy 2
Device# show event manager policy pending

no. job id status time of event             event type     name
1   1      pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
3   3      pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
```

Modifying the Scheduling Parameters of EEM Policy Events or Event Queues

To modify the scheduling parameters of the EEM policies, perform this task. The **show event manager policy pending** command displays policies that are assigned to the B or default class. All the currently pending policies are then changed to class A. After the configuration modification, the **show event manager policy pending** command shows all policies assigned as class A.

SUMMARY STEPS

1. **enable**
2. **show event manager policy pending**
3. **event manager scheduler modify** {**all** | **policy** *job-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**} | **class** *class-options*} [**queue-priority** {**high** | **last** | **low** | **normal**}][**processor** {**rp_primary** | **rp_standby**}]
4. **show event manager policy pending**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays the pending EEM policies. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.

	Command or Action	Purpose
Step 3	**event manager scheduler modify** {**all** | **policy** *job-id* | **queue-type** {**applet** | **call-home** | **axp** | **script**} | **class** *class-options*} [**queue-priority** {**high** | **last** | **low** | **normal**}][**processor** {**rp_primary** | **rp_standby**}] **Example:** `Device#` **event manager scheduler modify all class A**	Modifies the scheduling parameters of the EEM policies. • In this example, all currently pending EEM policies are assigned to class A.
Step 4	**show event manager policy pending** **Example:** `Device# show event manager policy pending`	Displays the EEM policies modified in Step 3 along with other pending policies. **Note** Only the syntax applicable to this task is used in this example. For more details, see the Cisco IOS Network Management Command Reference.

Examples

The following example shows how to modify the scheduling parameters of the EEM policies. In this example, the **show event manager policy pending** command displays policies that are assigned to the B or default class. All the currently pending policies are then changed to class A. After the configuration modification, the **show event manager policy pending** command verifies that all policies are now assigned as class A.

```
Device# show event manager policy pending
no. class   status time of event             event type     name
1   default pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2   default pend   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   B       pend   Thu Sep 7  02:54:04 2006  syslog         applet: three

Device# event manager scheduler modify all class A
Device# show event manager policy pending

no. class status time of event             event type     name
1   A     pend   Thu Sep 7  02:54:04 2006  syslog         applet: one
2   A     pend   Thu Sep 7  02:54:04 2006  syslog         applet: two
3   A     pend   Thu Sep 7  02:54:04 2006  syslog         applet: three
```

Verifying Class-Based Scheduled Activities of EEM Policies

To verify the scheduled activities of the EEM policies, use the **show event manager scheduler**command.

SUMMARY STEPS

1. **show event manager scheduler thread** [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]

DETAILED STEPS

show event manager scheduler thread [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]
This command displays all the EEM execution threads from the scheduler perspective and the details of the running policies. This command includes **detailed** and **queue-type** optional keywords. The following is sample output from this command:

Example:

```
Device# show event manager scheduler thread
1 Script threads service class default
  total: 1 running: 1 idle: 0
2 Script threads service class range A-D
  total: 3 running: 0 idle: 3
3 Applet threads service class default
  total: 32 running: 0 idle: 32
4 Applet threads service class W X
  total: 5 running: 0 idle: 5
```

To display the details of the running policies using the scheduler threads use the **detailed** keyword. The following is sample output for this keyword:

Example:

```
Device# show event manager scheduler thread detailed

1 Script threads service class default
total: 5 running: 5 idle: 0
1 job id: 12341, pid: 101, name: loop.tcl
2 job id: 12352, pid: 52, name: loop.tcl
3 job id: 12363, pid: 55, name: loop.tcl
4 job id: 12395, pid: 53, name: loop.tcl
5 job id: 12588, pid: 102, name: loop.tcl
2 Applet threads service class default
total: 32 running: 5 idle: 27
1 job id: 15585, pid: 104, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
2 job id: 15586, pid: 105, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
3 job id: 15587, pid: 106, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
4 job id: 15589, pid: 107, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
5 job id: 15590, pid: 80, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
```

To display the scheduler threads of a queue-type use the **queue-type** keyword. The following are the sample output for this keyword:

Example:

```
Device# show event manager sched thread queue-type applet
1 Applet threads service class default
total: 32 running: 7 idle: 25
Device# show event manager sched thread queue-type applet detailed
1 Applet threads service class default
total: 32 running: 5 idle: 27
1 job id: 15700, pid: 103, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
2 job id: 15701, pid: 104, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
3 job id: 15703, pid: 106, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
4 job id: 15704, pid: 107, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
5 job id: 15706, pid: 55, name: WDOG_SYSLG_CNTR_TRACK_INTF_APPL
```

Verifying Class-Based Active EEM Policies

To verify the active or the running EEM policies, use the **show event manager policy active** command.

SUMMARY STEPS

1. **show event manager policy active** [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]

DETAILED STEPS

show event manager policy active [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]
This command displays only the running EEM policies. This command includes **class**, **detailed** and **queue-type** optional keywords. The following is sample output from this command:

Example:

```
Device# show event manager policy active
no. job id p s status time of event event type name
1 12598 N A running Mon Oct29 20:49:37 2007 timer watchdog loop.tcl
2 12609 N A running Mon Oct29 20:49:42 2007 timer watchdog loop.tcl
3 12620 N A running Mon Oct29 20:49:46 2007 timer watchdog loop.tcl
4 12650 N A running Mon Oct29 20:49:59 2007 timer watchdog loop.tcl
5 12842 N A running Mon Oct29 20:51:13 2007 timer watchdog loop.tcl
default class - 6 applet events
no. job id p s status time of event event type name
1 15852 N A running Mon Oct29 21:11:09 2007 counter WDOG_SYSLG_CNTR_TRACK_INTF_APPL
2 15853 N A running Mon Oct29 21:11:09 2007 counter WDOG_SYSLG_CNTR_TRACK_INTF_APPL
3 15854 N A running Mon Oct29 21:11:10 2007 counter WDOG_SYSLG_CNTR_TRACK_INTF_APPL
4 15855 N A running Mon Oct29 21:11:10 2007 timer watchdog WDOG_SYSLG_CNTR_TRACK_INTF_APPL
5 15856 N A running Mon Oct29 21:11:11 2007 counter WDOG_SYSLG_CNTR_TRACK_INTF_APPL
6 15858 N A running Mon Oct29 21:11:11 2007 counter WDOG_SYSLG_CNTR_TRACK_INTF_APPL
```

Verifying Pending EEM Policies

To verify the EEM policies that are pending for execution, use the **show event manager policy pending** command. Use the optional keywords to specify EEM class-based scheduling options.

SUMMARY STEPS

1. **show event manager policy pending** [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]

DETAILED STEPS

show event manager policy pending [**queue-type** {**applet**| **call-home** | **axp** | **script**} **class** *class-options* | **detailed**]
This command displays only the pending policies. This command includes **class**, **detailed** and **queue-type** optional keywords. The following is sample output from this command:

Example:

```
Device# show event manager policy pending
no. job id p s status time of event event type name
1    12851   N A    pend    Mon Oct29   20:51:18 2007   timer watchdog   loop.tcl
2    12868   N A    pend    Mon Oct29   20:51:24 2007   timer watchdog   loop.tcl
3    12873   N A    pend    Mon Oct29   20:51:27 2007   timer watchdog   loop.tcl
4    12907   N A    pend    Mon Oct29   20:51:41 2007   timer watchdog   loop.tcl
5    13100   N A    pend    Mon Oct29   20:52:55 2007   timer watchdog   loop.tcl
```

Configuring EEM Applet (Interactive CLI) Support

The synchronous applets are enhanced to support interaction with the local console (tty) using two commands, **action gets** and **action puts**, and these commands allow users to enter and display input directly on the console. The output for synchronous applets will bypass the system logger. The local console will be opened by the applets and serviced by the corresponding synchronous Event Detector pty. Synchronous output will be directed to the opened console.

Reading and Writing Input from the Active Console for Synchronous EEM Applets

Use the following tasks to implement EEM applet interactive CLI support:

Reading Input from the Active Console

When a synchronous policy is triggered, the related console is stored in the publish information specification. The policy director will query this information in an event_reqinfo call, and store the given console information for use by the **action gets** command.

The **action gets** command reads a line of the input from the active console and stores the input in the variable. The trailing new line will not be returned.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event none**
5. **action** *label* **gets** *variable*
6. **action** *label* **syslog** [**priority** *priority-level* **msg** *msg-text*
7. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet action`	Registers the applet with the EEM and enters applet configuration mode.
Step 4	**event none** **Example:** `Device(config-applet)# event none`	Specifies that an EEM policy is to be registered with the EEM and can be run manually.
Step 5	**action** *label* **gets** *variable* **Example:** `Device(config-applet)# action label2 gets input`	Gets input from the local console in a synchronous applet and stores the value in the given variable when an EEM applet is triggered.
Step 6	**action** *label* **syslog** [**priority** *priority-level* **msg** *msg-text* **Example:** `Device(config-applet)# action label3 syslog msg “Input entered was \”$input\“”`	Specifies the action to be taken when an EEM applet is triggered. • In this example, the action to be taken is to write the value of the variable specified in Step 5, to syslog.
Step 7	**exit** **Example:** `Device(config-applet)# exit`	Exits applet configuration mode and returns to privileged EXEC mode.

Example

The following example shows how to get the input from the local tty in a synchronous applet and store the value

```
Device(config)# event manager applet action
Device(config-applet)# event none
Device(config-applet)# action label2 gets input
Device(config-applet)# action label3 syslog msg "Input entered was \"$input\""
```

Writing Input to the Active Console

When a synchronous policy is triggered, the related console is stored in the publish information specification. The policy director will query this information in an event_reqinfo call, and store the given console information for use by the **action puts** command.

The **action puts** command will write the string to the active console. A new line will be displayed unless the **nonewline** keyword is specified. The output from the **action puts** command for a synchronous applet is displayed directly to the console, bypassing the system logger. The output of the **action puts** command for an asynchronous applet is directed to the system logger.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event none**
5. **action** *label* **regexp** *string-pattern string-input* [*string-match* [*string-submatch1*] [*string-submatch2*] [*string-submatch3*]]
6. **action** *label* **puts** [**nonewline**] *string*
7. **exit**
8. **event manager run** *applet-name*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

	Command or Action	Purpose
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet action`	Registers the applet with the EEM and enters applet configuration mode.
Step 4	**event none** **Example:** `Device(config-applet)# event none`	Specifies that an EEM policy is to be registered with the EEM and can be run manually.
Step 5	**action** *label* **regexp** *string-pattern string-input* [*string-match* [*string-submatch1*] [*string-submatch2*] [*string-submatch3*]] **Example:** `Device(config-applet)# action 1 regexp "(.*) (.*) (.*)" "one two three" _match _sub1`	Specifies the action to match the regular expression pattern on an input string when an EEM applet is triggered.
Step 6	**action** *label* **puts** [**nonewline**] *string* **Example:** `Device(config-applet)# action 2 puts "match is $_match"`	Specifies the action of printing data directly to the local console when an EEM applet is triggered. • The **nonewline** keyword is optional and is used to suppress the display of the new line character.
Step 7	**exit** **Example:** `Device(config-applet)# exit`	Exits applet configuration mode and returns to privileged EXEC mode.
Step 8	**event manager run** *applet-name* **Example:** `Device# event manager run action`	Manually runs a registered EEM policy. • In this example, the policy registered in Step 3 is triggered and the associated actions specified in Step 5 and Step 6 are executed.

Example

The following example shows how the **action puts** command prints data directly to the local console:

```
Device(config-applet)# event manager applet puts
Device(config-applet)# event none
Device(config-applet)# action 1 regexp "(.*) (.*) (.*)" "one two three" _match _sub1
Device(config-applet)# action 2 puts "match is $_match"
Device(config-applet)# action 3 puts "submatch 1 is $_sub1"
Device# event manager run puts
```

```
match is one two three
submatch 1 is one
```

Configuring SNMP Library Extensions

Depending on your release, the SNMP Library Extensions feature allows you to perform the following configurations.

Prerequisites

To use this feature, you must be running Cisco IOS Release 12.4(22)T or a later release.

SNMP Get and Set Operations

The SNMP Library Extensions feature extends the EEM applet **action info** and Tcl **sys_reqinfo_snmp**commands to include functionality for SNMP get-one, get-next, getid and set-any operations.

SNMP Get Operation

The SNMP event manager performs the SNMP get operation to retrieve one or more variables for the managed objects. Using the **action info type snmp oid get-type**and **action info type snmp getid**commands, you can configure the SNMP event manager to send an SNMP get request by specifying the variables to retrieve, and the IP address of the agent.

For example, if you want to retrieve the variable with the OID value of 1.3.6.1.2.1.1.1, you should specify the variable value, that is 1.3.6.1.2.1.1.1. If the specified values do not match, a trap will be generated and an error message will be written to the syslog history.

The **action info type snmp oid get-type** command specifies the type of the get operation to be performed. To retrieve the exact variable, the get operation type should be specified as **exact**. To retrieve a lexicographical successor of the specified OID value, the get operation type should be set to **next**.

The table below shows the built-in variables, in which the values retrieved from SNMP get operation are stored.

Table 7: Built-in Variables for action info type snmp oid Command

Built-in Variable	Description
_info_snmp_oid	The SNMP object ID.
_info_snmp_value	The value string of the associated SNMP data element.

GetID Operation

The **action info type snmp getid** command retrieves the following variables from the SNMP entity:

- sysDescr.0

- sysObjectID.0
- sysUpTime.0
- sysContact.0
- sysName.0
- sysLocation.0

The table below shows the built-in variables, in which the values retrieved from the SNMP getID operation are stored.

Table 8: Built-in Variables for action info type snmp getid Command

Built-in Variable	Description
_info_snmp_syslocation_oid	The OID value of the sysLocation variable.
_info_snmp_syslocation_value	The value string for the sysLocation variable.
_info_snmp_sysdescr_oid	The OID value of the sysDescr variable.
_info_snmp_sysdescr_value	The value string for the sysDescr variable.
_info_snmp_sysobjectid_oid	The OID value of the sysObjectID variable.
_info_snmp_sysobjectid_value	The value string for the sysObjectID variable.
_info_snmp_sysuptime_oid	The OID value of the sysUptime variable.
_info_snmp_sysuptime_value	The value string for the sysUptime variable.
_info_snmp_syscontact_oid	The OID value of the sysContact variable.
_info_snmp_syscontact_value	The value string for the sysContact variable.

The get operation requests can be sent to both local and remote hosts.

SNMP Set Operation

All SNMP variables are assigned a default value in the MIB view. The SNMP event manager can modify the value of these MIB variables through set operation. The set operation can be performed only on the system that allows read-write access.

To perform a set operation, you must specify the type of the variable and the value associated with it.

The table below shows the valid OID types and values for each OID type.

Table 9: OID Type and Value for Set Operation

OID Type	Description
counter32	A 32-bit number with a minimum value of 0. When the maximum value is reached, the counter resets to 0. Integer value in the range from 0 to 4294967295 is valid.
gauge	A 32-bit number with a minimum value of 0. For example, the interface speed on a device is measured using a gauge object type. Integer value in the range from 0 to 4294967295 is valid.
integer	A 32-bit number used to specify a numbered type within the context of a managed object. For example, to set the operational status of a device interface, 1 represents up and 2 represents down. Integer value in the range from 0 to 4294967295 is valid.
ipv4	IP version 4 address. IPv4 address in dotted decimal notation is valid.
octet string	An octet string in hexadecimal notation used to represent physical addresses. Text strings are valid.
string	An octet string in text notation used to represent text strings. Text strings are valid.
unsigned32	A 32-bit number used to represent decimal value.Unsigned integer value in the range from 0 to 4294967295 is valid.

The set operation can carried out on both local and remote hosts.

SNMP Traps and Inform Requests

Traps are SNMP notifications that alert the SNMP manager or the NMS to a network condition.

SNMP inform requests refer to the SNMP notifications that alert the SNMP manager to a network condition and request for confirmation of receipt from the SNMP manager.

An SNMP event occurs when SNMP MIB object ID values are sampled, or when the SNMP counter crosses a defined threshold. If the notifications are enabled and configured for such events, the SNMP traps or inform messages generated. An SNMP notification event is triggered when an SNMP trap or inform message is received by the event manager server.

To send an SNMP trap or inform message when an Embedded Event Manager (EEM) applet is triggered, the **action info type snmp trap** and **action info type snmp inform**commands are used. The CISCO-EMBEDDED-EVENT-MGR-MIB.my is used to define the trap and inform messages.

Configuring EEM Applet for SNMP Get and Set Operations

While registering a policy with the event manager server, the actions associated with an SNMP event can be configured.

Perform this task to configure EEM applet for SNMP set and get operations.

Before You Begin

- SNMP event manager must be configured using the **snmp-server manager**command.
- The SNMP community string should be set by using the **snmp-server community** command to enable access to the SNMP entity.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. Do one of the following:
    - **event snmp oid** *oid-value* **get-type** {**exact** | **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb** | **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value*
5. **action label info type snmp oid** *oid-value* **get-type** {**exact**| **next**} [**community** *community-string*] [**ipaddr** *ip-address*]
6. **action label info type snmp oid** *oid-value* **set-type** *oid-type* *oid-type-value* **community** *community-string* [**ipaddr** *ip-address*]
7. **action label info type snmp getid** *oid-value* [**community** *community-string*] [**ipaddr** *ip-address*]
8. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

	Command or Action	Purpose
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet snmp`	Registers the applet with the event manager server and enters applet configuration mode.

Step 4	Do one of the following:
	• **event snmp oid** *oid-value* **get-type** {**exact** | **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb** | **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value*
	
	Example:
	`Device(config-applet)# event snmp oid`
	
	Example:
	`1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact`
	
	Example:
	`entry-op lt entry-val 5120000 poll-interval 90`
		Specifies the event criteria that cause the EEM applet to run.
		• In this example, an EEM event is triggered when free memory falls below the value of 5120000.
		• Exit criteria are optional, and if not specified, event monitoring is reenabled immediately.

Step 5	**action label info type snmp oid** *oid-value* **get-type** {**exact**| **next**} [**community** *community-string*] [**ipaddr** *ip-address*]
	
	Example:
	`Device(config-applet)# action 1.3 info type`
	
	Example:
	`snmp oid 1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type`
	
	Example:
	`exact community public ipaddr 172.17.16.69`
		Specifies the type of get operation to perform.
		• In this example, the type of get operation is specified as exact and community string is specified as public.

Step 6	**action label info type snmp oid** *oid-value* **set-type** *oid-type oid-type-value* **community** *community-string* [**ipaddr** *ip-address*] **Example:** `Device(config-applet)# action 1.4 info type`	(Optional) Specifies the variable to be set. • In this example, the sysName.0 variable is specified for the set operation and community string is specified as rw. **Note** For set operation, you must specify the SNMP community string.

	Command or Action	Purpose

	Example:
	`snmp oid 1.3.6.1.4.1.9.9.48.1.1.1.6.1 set-type`
	Example:
	`integer 42220 sysName.0 community rw ipaddr`
	Example:
	`172.17.16.69`	

Step 7	**action label info type snmp getid** *oid-value* [**community** *community-string*] [**ipaddr** *ip-address*]
	Example:
	`Device(config-applet)# action 1.3 info type`
	Example:
	`snmp getid community public ipaddr 172.17.16.69`
		(Optional) Specifies if the individual variables should be retrieved by the getid operation.

Step 8	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Configuring EEM Applet for SNMP OID Notifications

Perform this task to configure SNMP notifications.

Before You Begin

- SNMP event manager must be configured using the **snmp-server manager**command and SNMP agents must be configured to send and receive SNMP traps generated for an EEM policy.
- SNMP traps and informs must be enabled by using the **snmp-server enable traps event-manager** and **snmp-server enable traps** commands, to allow traps and inform requests to be sent from the device to the event manager server.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. Do one of the following:
   - **event snmp oid** *oid-value* **get-type** {**exact** | **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb** | **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value*
5. **action label info type snmp var** *variable-name* **oid** *oid-value* *oid-type oid-type-value*
6. **action label info type snmp trap enterprise-oid** *enterprise-oid-value* **generic-trapnum** *generic-trap-number* **specific-trapnum** *specific-trap-number* **trap-oid** *trap-oid-value* **trap-var** *trap-variable*
7. **action label info type snmp inform trap-oid** *trap-oid-value* **trap-var** *trap-variable* **community** *community-string* **ipaddr** *ip-address*
8. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet snmp`	Registers the applet with the event manager server and enters applet configuration mode.
Step 4	Do one of the following: • **event snmp oid** *oid-value* **get-type** {**exact** | **next**} **entry-op** *operator* **entry-val** *entry-value*[**exit-comb** | **and**}] [**exit-op** *operator*] [**exit-val** *exit-value*] [**exit-time** *exit-time-value*] **poll-interval** *poll-int-value*	Specifies the event criteria that cause the EEM applet to run. • In this example, an EEM event is triggered when free memory falls below the value of 5120000. • Exit criteria are optional, and if not specified, event monitoring is reenabled immediately.

	Command or Action	Purpose

	Example:
	`Device(config-applet)# event snmp oid`
	Example:
	`1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact`
	Example:
	`entry-op lt entry-val 5120000 poll-interval 90`	

Step 5	**action label info type snmp var** *variable-name* **oid** *oid-value* *oid-type oid-type-value*
	Example:
	`Device(config-applet)# action 1.3 info type`
	Example:
	`snmp var sysDescr.0 oid`
	Example:
	`1.3.6.1.4.1.9.9.48.1.1.1.6.1 integer 4220`
		Specifies the instance of a managed object and its value.
		• In this example, the sysDescr.0 variable is used.

Step 6	**action label info type snmp trap enterprise-oid** *enterprise-oid-value* **generic-trapnum** *generic-trap-number* **specific-trapnum** *specific-trap-number* **trap-oid** *trap-oid-value* **trap-var** *trap-variable*
	Example:
	`Device(config-applet)# action 1.4 info type`
	Example:
	`snmp trap enterprise-oid 1.3.6.1.4.1.1`
	Example:
	`generic-trapnum 4 specific-trapnum 7 trap-oid`
	Example:
	`1.3.6.1.4.1.1.226.0.2.1 trap-var sysUpTime.0`
		Generates an SNMP trap when the EEM applet is triggered.
		• In this example, the authenticationFailure trap is generated.
		Note The specific trap number refers to the enterprise-specific trap, which is generated when an enterprise event occurs. If the generic trap number is not set to 6, the specific trap number you specify will be used to generate traps.

	Command or Action	Purpose

Step 7	**action label info type snmp inform trap-oid** *trap-oid-value* **trap-var** *trap-variable* **community** *community-string* **ipaddr** *ip-address*
	
	Example:
	`Device(config-applet)# action 1.4 info type`
	
	Example:
	`snmp inform trap-oid 1.3.6.1.4.1.1.226.0.2.1`
	
	Example:
	`trap-var sysUpTime.0 community public ipaddr`
	
	Example:
	`172.69.16.2`
		Generates an SNMP inform request when the EEM applet is triggered.
		• In this example, the inform request is generated for the sysUpTime.0 variable.

Step 8	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged mode.

Configuring Variable Logic for EEM Applets

The Variable Logic for EEM Applets feature adds the ability to apply conditional logic within EEM applets. Before variable logic is introduced, applets have a linear structure where each action is executed in the order in which they are configured when the event is triggered. Conditional logic introduces a control structure that can change the flow of actions within applets depending on conditional expressions. Each control structure can contain a list of applet actions including looping and if/else actions which determine if the structure is executed or not.

The information in applet configuration mode is presented as background to set the context for the action commands.

To provide a consistent user interface between the Tool Command Language (Tcl) and the applet (CLI) based EEM policies, the following criteria are followed:

- Event specification criteria are written in Tcl in the Tcl based implementation.
- Event specification data is written using the CLI applet submode configuration statements in the applet-based implementation.

Applet configuration mode is entered using the event manager applet command. In applet configuration mode the config prompt changes to (config-applet)#. In applet configuration mode two types of config statements are supported:

- event - used to specify the event criteria to cause this applet to run.
- action - used to specify a built-in action to perform.

Multiple **action** applet config commands are allowed within an applet configuration. If no **action** applet config command is present, a warning is displayed, upon exit, stating no statements are associated with this applet. When no statements are associated with this applet, events get triggered but no action is taken. If no commands are specified in applet configuration mode, the applet will be removed upon exit. The exit applet config command is used to exit from applet configuration mode.

Depending on your release, the Variable Logic for EEM Applets feature allows you to perform the following configurations.

Prerequisites

To use this feature, you must be running Cisco IOS Release 12.4(22)T or a later release.

Configuring Variable Logic for EEM Applets

EEM 3.0 adds new applet action commands to permit simple variable logic within applets.

To configure the variable logic using action commands perform the following tasks.

Specifying a Loop of Conditional Blocks

To specify a loop of a conditional block when an EEM applet is triggered, perform this task. In this task, a conditional loop is set to check if the value of the variable is less than 10. If the value of the variable is less than 10, then the message ‘i is $_i’ is written to the syslog.

Note Depending on your release, the **set** (EEM) command is replaced by the **action set** command. See the **action** *label* **set** command for more information. If the set (EEM) command is entered in certain releases, the IOS parser translates the **set** command to the **action** *label* **set** command.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **action** *label* **set**
5. **action** *label* **while** *string_op1 operator string_op2*
6. Add any action as required.
7. **action** *label* **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet condition`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**action** *label* **set** **Example:** `Device(config-applet)#`**action 1.0 set i 2**	Sets an action for the event. • In this example, the value of the variable i is set to 2.
Step 5	**action** *label* **while** *string_op1 operator string_op2* **Example:** `Device(config-applet)# action 2 while $i lt 10`	Specifies a loop of a conditional block. • In this example, a loop is set to check if the value of the variable i is less than 10.
Step 6	Add any action as required. **Example:** `Device(config-applet)#` **action 3 syslog msg "i is $i"**	Performs the action as indicated by the action command. • In this example, the message 'i is $_i' is written to the syslog.
Step 7	**action** *label* **end** **Example:** `Device(config-applet)#` **action 3 end**	Exits from the running action.

Specifying if else Conditional Blocks

To specify the beginning of an if conditional statement followed by an else conditional statement, perform this task. The if or else conditional statements can be used in conjunction with each other or separately. In this task, the value of a variable is set to 5. An if conditional block is then specified to check if the value of the variable is less than 10. Provided the if conditional block is satisfied, an action command to output the message ‘x is less than 10’ is specified.

Following the if conditional block, an else conditional block is specified. Provided the if conditional block is not satisfied, an action command to output the message ‘x is greater than 10’ is specified.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **action** *label* **set** *variable-name variable-value*
5. **action** *label* **if** [*stringop1*] {**eq** | **gt** | **ge** | **lt** | **le** | **ne**} [*stringop2*]
6. Add any action as required.
7. **action** *label* **else**
8. Add any action as required.
9. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet ifcondition`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**action** *label* **set** *variable-name variable-value*	Sets an action for the event.

	Command or Action	Purpose
	Example: `Device(config-applet)# action 1.0 set x 5`	• In this example, the value of the variable x is set to 5.
Step 5	**action** *label* **if** [*stringop1*] {**eq** | **gt** | **ge** | **lt** | **le** | **ne**} [*stringop2*] **Example:** `Device(config-applet)# action 2.0 if $x lt 10`	Specifies an if conditional statement. • In this example, an if conditional statement to check if the value of the variable is less than 10.
Step 6	Add any action as required. **Example:** `Device(config-applet)# action 3.0 puts "$x is less than 10"`	Performs the action as indicated by the action command. • In this example, the message '5 is less than 10' is displayed on the screen.
Step 7	**action** *label* **else** **Example:** `Device(config-applet)# action 4.0 else`	Specifies an else conditional statement
Step 8	Add any action as required. **Example:** `Device(config-applet)# action 5.0`	Performs the action as indicated by the action command. • In this example, the message '5 is greater than 10' is displayed on the screen.
Step 9	**end** **Example:** `Device(config-applet)# end`	Exits from the running action.

Specifying foreach Iterating Statements

To specify a conditional statement that iterates over an input string using the delimiter as a tokenizing pattern, perform this task. The foreach iteration statement is used to iterate through a collection to get the desired information. The delimiter is a regular expression pattern string. The token found in each iteration is assigned to the given iterator variable. All arithmetic calculations are performed as long integers with out any checks for overflow. In this task, the value of the variable x is set to 5. An iteration statement is set to run through the input string red, blue, green, orange. For every element in the input string, a corresponding message is displayed on the screen.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **action** *label* **foreach** [*string-iterator*] [*string-input*] [*string-delimiter*]
5. Specify any action command
6. **action** *label* **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet iteration`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**action** *label* **foreach** [*string-iterator*] [*string-input*] [*string-delimiter*] **Example:** `Device(config-applet)# action 2.0 foreach iterator "red blue green orange"`	Iterates over an input string using the delimter as a tokenizing pattern. • In this example, the iteration is run through the elements of the input string - red, blue, green and orange.
Step 5	Specify any action command **Example:** `Device(config-applet)# action 3.0 puts "Iterator is $iterator"`	Performs the action as indicated by the action command. • In this example, the following message is displayed on the screen: Iterator is red Iterator is blue Iterator is green Iterator is orange

	Command or Action	Purpose
Step 6	**action** *label* **end** **Example:** `Device(config-applet)# action 4.0 end`	Exits from the running action.

Using Regular Expressions

To match a regular expression pattern with an input string, perform this task. Using regular expressions, you can specify the rules for a set of possible strings to be matched.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **action** *label* **regexp** *string-pattern string-input* [*string-match* [*string-submatch1*] [*string-submatch2*] [*string-submatch3*]]

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#` **configure terminal**	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)#` **event manager applet regexp**	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**action** *label* **regexp** *string-pattern string-input* [*string-match* [*string-submatch1*] [*string-submatch2*] [*string-submatch3*]]	Specifies an expression pattern to match with an input string. • In this example, an input string of 'red blue green' is specified. When the expression pattern matches the input

	Command or Action	Purpose
	Example: `Device(config-applet)#` **action 2.0 regexp "(.*) (.*) (.*)" "red blue green" _match _sub1**	string, the entire result **red blue green** is stored in the variable **_match** and the submatch **red**is stored in the variable **_sub1**.

Incrementing the Values of Variables

To increment the value of variables, perform this task. In this task, the value of a variable is set to 20 and then the value is incremented by 12.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **action** *label* **set**
5. **action** *label* **increment** *variable-name long-integer*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet increment`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.
Step 4	**action** *label* **set**	Sets an action for the event.

	Command or Action	Purpose
	Example: `Device(config-applet)#`**action 1.0 set varname 20**	• In this example, the value of the variable is set to 20.
Step 5	**action** *label* **increment** *variable-name long-integer* **Example:** `Device(config-applet)#`**action 2.0 increment varname 12**	Increments the value of variable by the specified long integer. • In this example, the value of the variable is incremented by 12.

Configuring Event SNMP Object

Perform this task to register the Simple Network Management Protocol (SNMP) object event for an Embedded Event Manager (EEM) applet that is run by sampling SNMP object.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **event snmp-object oid** *oid-value* **type** *value* **sync** {**yes** | **no**} **skip** {**yes** | **no**} **istable** {**yes** | **no**} [**default** *seconds*] [**maxrun** *maxruntime-number*]
5. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

	Command or Action	Purpose
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet manual-policy`	Registers the applet with the Embedded Event Manager and enters applet configuration mode.

Step 4	**event snmp-object oid** *oid-value* **type** *value* **sync** {**yes** | **no**} **skip** {**yes** | **no**} **istable** {**yes** | **no**} [**default** *seconds*] [**maxrun** *maxruntime-number*]
	
	Example:
	
	`Device(config-applet)# event snmp-object oid 1.9.9.9.9 type gauge sync yes`
	
	Example:
	
	`action 1 syslog msg "oid = $_snmp_oid"`
	
	Example:
	
	`action 2 syslog msg "request = $_snmp_request"`
	
	Example:
	
	`action 3 syslog msg "request_type = $_snmp_request_type"`
		Registers the Simple Network Management Protocol (SNMP) object event for an Embedded Event Manager (EEM) applet to intercept SNMP GET and SET requests for an object.
		
		The default for this command is that it is not configured. If this command is configured the defaults are the same as in the description of the syntax options,
		
		• The **oid**keyword specifies the SNMP object identifier (object ID).
		
		• The *oid-value* argument can be the Object ID value of the data element, in SNMP dotted notation. An OID is defined as a type in the associated MIB, CISCO-EMBEDDED-EVENT-MGR-MIB, and each type has an object value.
		
		• The **istable** keyword specifies whether the OID is an SNMP table.
		
		• The sync keyword specifies that the applet is to run in synchronous mode. The return code from the applet indicates whether to reply to the SNMP request. The description for code 0 is “do not reply to the request” and the description for code 1 is “reply to the request”. When the return code from the applet replies to the request, a value is specified in the applet for the object using **action snmp-object-value** command.
		
		• The **type**keyword specifies the type of object.
		
		• The *value* argument is the value of the object.
		
		• Theskip keyword specifies whether to skip CLI command execution.
		
		• The **default** keyword specifies the time to process the SET or GET request normally by the applet. If the **default** keyword is not specified, the default time period is set to 30 seconds.
		
		• The *milliseconds* argument is the time period during which the SNMP Object event detector waits for the policy to exit.
		
		• The **maxrun** keyword specifies the maximum runtime of the applet. If the **maxrun** keyword is specified, the *maxruntime-number* value must be specified. If the **maxrun** keyword is not specified, the default applet run time is 20 seconds.
		
		• The *milliseconds* argument is the maximum runtime of the apple in milliseconds. If the argument is not specified, the default 20-second run-time limit is used.

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

Disabling AAA Authorization

Perform this task to allow EEM policies to bypass AAA authorization when triggered.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name* [**authorization bypass**] [**class** *class-options*] [**trap**]
4. **exit**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**event manager applet** *applet-name* [**authorization bypass**] [**class** *class-options*] [**trap**] **Example:** `Device(config)# event manager applet one class A authorization bypass`	Registers the applet with the Embedded Event Manager (EEM) and enters applet configuration mode.

	Command or Action	Purpose
Step 4	**exit** **Example:** `Device(config-aaplet)# exit`	Exits device configuration applet mode and returns to privileged EXEC mode.

Configuring Description of an Embedded Event Manager Applet

Perform this task to describe an EEM applet. The description of an applet can be added in any order, before or after any other applet configuration. Configuring a new description for an applet that already has a description overwrites the current description. An applet description is optional.

Perform this task to configure a new description for an applet.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **event manager applet** *applet-name*
4. **description** *line*
5. **event syslog pattern** *regular-expression*
6. **action** *label* **syslog msg** *msg-text*
7. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

	Command or Action	Purpose
Step 3	**event manager applet** *applet-name* **Example:** `Device(config)# event manager applet increment`	Registers the applet with the EEM and enters applet configuration mode.
Step 4	**description** *line* **Example:** `Device(config-applet)# description "This applet looks for the word count in syslog messages"`	Adds or modifies the description of an EEM applet that is run by sampling Simple Network Management Protocol (SNMP).
Step 5	**event syslog pattern** *regular-expression* **Example:** `Device(config-applet)# event syslog pattern "count"`	Specifies the event criteria for an Embedded Event Manager (EEM) applet that is run by matching syslog messages.
Step 6	**action** *label* **syslog msg** *msg-text* **Example:** `Device(config-applet)# action 1 syslog msg hi`	Specifies the action to be taken when an EEM applet is triggered. • In this example, the action taken is to write a message to syslog. • The *msg-text* argument can be character text, an environment variable, or a combination of the two.
Step 7	**end** **Example:** `Device(config-applet)# end`	Exits applet configuration mode and returns to privileged EXEC mode.

Configuration Examples for Writing EEM Policies Using the Cisco IOS CLI

Embedded Event Manager Applet Configuration Examples

The following examples show how to create an EEM applet for some of the EEM event detectors. These examples follow steps outlined in the Registering and Defining an Embedded Event Manager Applet, on page 37.

Application-Specific Event Detector

The following example shows how a policy named EventPublish_A runs every 20 seconds and publishes an event type numbered 1 to an EEM subsystem numbered 798. The subsystem value of 798 specifies that a publish event has occurred from an EEM policy. A second policy named EventPublish_B is registered to run when the EEM event type 1 occurs with subsystem 798. When the EventPublish_B policy runs, it sends a message to syslog containing data passed as an argument from the EventPublish_A policy.

```
event manager applet EventPublish_A
 event timer watchdog time 20.0
 action 1.0 syslog msg "Applet EventPublish_A"
 action 2.0 publish-event sub-system 798 type 1 arg1 twenty
 exit
event manager applet EventPublish_B
 event application sub-system 798 type 1
 action 1.0 syslog msg "Applet EventPublish_B arg1 $_application_data1"
```

CLI Event Detector

The following example shows how to specify an EEM applet to run when the Cisco IOS **write memory** CLI command is run. The applet provides a notification that this event has occurred via a syslog message. In the example, the **sync** keyword is configured with the yes argument, and this means that the event detector is notified when this policy completes running. The exit status of the policy determines whether the CLI command will be executed. In this example, the policy exit status is set to one and the CLI command runs.

```
event manager applet cli-match
 event cli pattern "write mem.*" sync yes
 action 1.0 syslog msg "$_cli_msg Command Executed"
 set 2.0 _exit_status 1
```

The following example shows an applet which matches the **cli pattern** with the test argument. When **show access-list test** is entered, the CLI event detector matches the test argument, and the applet is triggered. The **debug event manager detector cli** output is added to show num_matches is set to one.

```
!
event manager applet EEM-PIPE-TEST
 event cli pattern "test" sync yes
 action 1.0 syslog msg "Pattern matched!"
!
*Aug 23 23:19:59.827: check_eem_cli_policy_handler: command_string=show access-lists test
*Aug 23 23:19:59.827: check_eem_cli_policy_handler: num_matches = 1, response_code = 4
*Aug 23 23:19:59.843: %HA_EM-6-LOG: EEM-PIPE-TEST: Pattern matched!
```

Note The functionality provided in the CLI event detector only allows a regular expression pattern match on a valid IOS CLI command itself. This does not include text after a pipe (|) character when redirection is used.

The following example shows that when **show version | include test** is entered, the applet fails to trigger because the CLI event detector does not match on characters entered after the pipe (|) character and the **debug event manager detector cli** output shows num_matches is set to zero.

```
*Aug 23 23:20:16.827: check_eem_cli_policy_handler: command_string=show version
*Aug 23 23:20:16.827: check_eem_cli_policy_handler: num_matches = 0, response_code = 1
```

Counter Event Detector and Timer Event Detector

The following example shows that the EventCounter_A policy is configured to run once a minute and to increment a well-known counter called critical_errors. A second policy--EventCounter_B--is registered to be

triggered when the well-known counter called critical_errors exceeds a threshold of 3. When the EventCounter_B policy runs, it resets the counter to 0.

```
event manager applet EventCounter_A
 event timer watchdog time 60.0
 action 1.0 syslog msg “EventCounter_A”
 action 2.0 counter name critical_errors op inc value 1
 exit
event manager applet EventCounter_B
 event counter name critical_errors entry-op gt entry-val 3 exit-op lt exit-val 3
 action 1.0 syslog msg “EventCounter_B”
 action 2.0 counter name critical_errors op set value 0
```

Interface Counter Event Detector

The following example shows how a policy named EventInterface is triggered every time the receive_throttle counter for Fast Ethernet interface 0/0 is incremented by 5. The polling interval to check the counter is specified to run once every 90 seconds.

```
event manager applet EventInterface
 event interface name FastEthernet0/0 parameter receive_throttle entry-op ge entry-val 5
 entry-val-is-increment true poll-interval 90
 action 1.0 syslog msg "Applet EventInterface"
```

Resource Event Detector

The following example shows how to specify event criteria based on an ERM event report for a policy defined to report high CPU usage:

```
event manager applet policy-one
 event resource policy cpu-high
 action 1.0 syslog msg “CPU high at $_resource_current_value percent”
```

RF Event Detector

The RF event detector is only available on networking devices that contain dual Route Processors (RPs). The following example shows how to specify event criteria based on an RF state change notification:

```
event manager applet start-rf
 event rf event rf_prog_initialization
 action 1.0 syslog msg "rf state rf_prog_initialization reached"
```

RPC Event Detector

The RPC event detector allows an outside entity to make a Simple Object Access Protocol (SOAP) request to the device and invokes a defined EEM policy or script. The following example shows how an EEM applet called Event_RPC is being registered to run an EEM script:

```
event manager applet Event_RPC
 event rpc
 action print puts "hello there"
```

The following example shows the format of the SOAP request and reply message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://www.cisco.com/eem.xsd">
  <SOAP:Body>
    <run_eemscript>
      <script_name>Event_RPC</script_name>
    </run_eemscript>
  </SOAP:Body>
```

```
</SOAP:Envelope>
]]>]]>
<?xml version="1.0" encoding="UTF-8"?><SOAP:Envelope
xmlns:SOAP="http://www.cisco.com/eem.xsd"><SOAP:Body>
<run_eemscript_response><return_code>0</return_code><output></output></run_eemscript_response></SOAP:Body></SOAP:Envelope>]]>]]>
```

SNMP Event Detector

The following example shows how to specify an EEM applet to run when the CPU usage is greater than 75 percent. When the EEM applet runs, the CLI commands **enable** and **show cpu processes** are run, and an e-mail containing the result of the **show cpu processes** command is sent to an engineer.

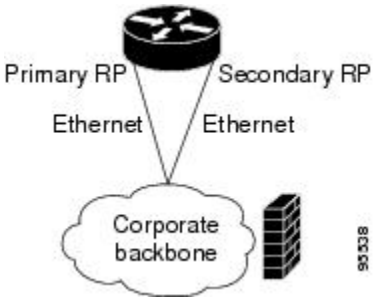
```
event manager applet snmpcpuge75
 event snmp oid 1.3.6.1.4.1.9.9.109.1.1.1.1.3.1 get-type exact entry-op ge entry-val 75
poll-interval 10
 action 1.0 cli command "enable"
 action 2.0 cli command "show process cpu"
 action 3.0 mail server "192.168.1.146" to "engineer@cisco.com" from "devtest@cisco.com"
subject "B25 PBX Alert" body "$_cli_result"
```

The next example is more complex and shows how to configure an EEM applet that causes a switch to the secondary (redundant) Route Processor (RP) when the primary RP runs low on memory.

This example illustrates a method for taking preventative action against a software fault that causes a memory leak. The action taken here is designed to reduce downtime by switching over to a redundant RP when a possible memory leak is detected.

The figure below shows a dual RP device that is running an EEM image. An EEM applet has been registered through the CLI using the **event manager applet** command. The applet will run when the available memory on the primary RP falls below the specified threshold of 5,120,000 bytes. The applet actions are to write a message to syslog that indicates the number of bytes of memory available and to switch to the secondary RP.

Figure 2: Dual RP Topology



The commands used to register the policy are shown below.

```
event manager applet memory-demo
 event snmp oid 1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact entry-op lt entry-val 5120000
poll-interval 90
 action 1.0 syslog priority critical msg "Memory exhausted; current available memory is
$_snmp_oid_val bytes"
 action 2.0 force-switchover
```

The registered applet is displayed using the **show event manager policy registered** command:

```
Device# show event manager policy registered
No.  Type    Event Type          Time Registered           Name
1    applet  snmp                Thu Jan30  05:57:16 2003  memory-demo
 oid {1.3.6.1.4.1.9.9.48.1.1.1.6.1} get-type exact entry-op lt entry-val {5120000}
poll-interval 90
```

```
 action 1.0 syslog priority critical msg "Memory exhausted; current available memory is
$_snmp_oid_val bytes"
 action 2.0 force-switchover
```

For the purpose of this example, a memory depletion is forced on the device, and a series of **show memory** commands are executed to watch the memory deplete:

```
Device# show memory
                Head    Total(b)     Used(b)     Free(b)   Lowest(b)  Largest(b)
Processor   53585260   212348444   119523060    92825384    92825384    92365916
Fast        53565260      131080       70360       60720       60720       60668
Device# show memory
                Head    Total(b)     Used(b)     Free(b)   Lowest(b)  Largest(b)
Processor   53585260   212364664   164509492    47855172    47855172    47169340
Fast        53565260      131080       70360       60720       60720       60668
Device# show memory
                Head    Total(b)     Used(b)     Free(b)   Lowest(b)  Largest(b)
Processor   53585260   212369492   179488300    32881192    32881192    32127556
Fast        53565260      131080       70360       60720       60720       60668
```

When the threshold is reached, an EEM event is triggered. The applet named memory-demo runs, causing a syslog message to be written to the console and a switch to be made to the secondary RP. The following messages are logged:

```
00:08:31: %HA_EM-2-LOG: memory-demo: Memory exhausted; current available memory is
4484196 bytes
00:08:31: %HA_EM-6-FMS_SWITCH_HARDWARE: fh_io_msg: Policy has requested a hardware
switchover
```

The following is partial output from the **show running-config** command on both the primary RP and the secondary (redundant) RP:

```
redundancy
 mode sso
.
.
!
event manager applet memory-demo
 event snmp oid 1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact entry-op lt entry-val
5120000 poll-interval 90
 action 1.0 syslog priority critical msg "Memory exhausted; current available memory
is $_snmp_oid_val bytes"
 action 2.0 force-switchover
```

SNMP Notification Event Detector

The following example shows how to configure the **snmp-server community** public RW and **snmp-server manager** commands before **event snmp-notification** is configured.

```
snmp-server community public RW
 snmp-server manager
```

The following example shows how an EEM applet called SNMP_Notification is being registered to run an EEM script when the device receives an SNMP notification on destination IP address 192.168.1.1 for object ID 1 whose value equals 10.

```
event manager applet SNMP_Notification
 event snmp-notification dest_ip_address 192.168.1.1 oid 1 op eq oid-value 10
 action 1 policy eem_script
```

Syslog Event Detector

The following example shows how to specify an EEM applet to run when syslog identifies that Ethernet interface 1/0 is down. The applet sends a message about the interface to syslog.

```
event manager applet interface-down
```

```
event syslog pattern ".*UPDOWN.*Ethernet1/0.*" occurs 4
action 1.0 syslog msg "Ethernet interface 1/0 changed state 4 times"
```

Configuration Examples for Embedded Event Manager Applet

Example Identity Event Detector

The following example shows how a policy named "EventIdentity" is triggered every time the authentication on the Fast Ethernet interface 0 is success.

```
event manager applet EventIdentity
 event identity interface FastEthernet0 authc success
action 1.0 syslog msg "Applet EventIdentity"
```

Example MAT Event Detector

The following example shows how a policy named "EventMat" is triggered every time a mac-address is learned in the mac-address-table.

```
event manager applet EventMat
 event mat interface FastEthernet0
action 1.0 syslog msg "Applet EventMat"
```

Example Neighbor-Discovery Event Detector

The following example shows how a policy named "EventNeighbor" is triggered when a Cisco Discovery Protocol (CDP) cache entry changes.

```
event manager applet EventNeighbor
 event neighbor-discovery interface FastEthernet0 cdp all
action 1.0 syslog msg "Applet EventNeighbor"
```

Embedded Event Manager Manual Policy Execution Examples

The following examples show how to use the none event detector to configure an EEM policy (applet or script) to be run manually.

Using the event manager run Command

This example shows how to run a policy manually using the **event manager run** command. The policy is registered using the **event none** command under applet configuration mode and then run from global configuration mode using the **event manager run** command.

```
event manager applet manual-policy
 event none
 action 1.0 syslog msg "Manual-policy triggered"
 end
!
event manager run manual-policy
```

Using the action policy Command

This example shows how to run a policy manually using the **action policy** command. The policy is registered using the **event none** command under applet configuration mode, and then the policy is executed using the **action policy** command in applet configuration mode.

```
event manager applet manual-policy
 event none
 action 1.0 syslog msg "Manual-policy triggered"
 exit
!
event manager applet manual-policy-two
 event none
 action 1.0 policy manual-policy
 end
!
event manager run manual-policy-two
```

Configuring and Tracking a Stub Object Using Embedded Event Manager Example

This example shows how to create a stub object, set the state of the stub object, and configure an EEM applet to be run when the tracked object changes. The enhanced object tracking (EOT) event detector is used, and actions are specified to both set and read the state of the object. This example allows EEM to define an EOT object that may be manipulated by other EOT clients. An EEM policy can be a trigger for any EOT object including objects defined for other EOT clients or for an object defined by EEM.

```
track 10 stub-object
 default-state down
!
event manager applet track-ten
 event track 10 state any
 action 1.0 track set 10 state up
 action 2.0 track read 10
```

Embedded Event Manager Watchdog System Monitor (Cisco IOS) Event Detector Configuration Example

The following example shows how to configure three EEM applets to demonstrate how the Cisco IOS watchdog system monitor (IOSWDSysMon) event detector works.

Watchdog System Monitor Sample1 Policy

The first policy triggers an applet when the average CPU usage for the process named IP Input is greater than or equal to 1 percent for 10 seconds:

```
event manager applet IOSWD_Sample1
 event ioswdsysmon sub1 cpu-proc taskname "IP Input" op ge val 1 period 10
 action 1.0 syslog msg "IOSWD_Sample1 Policy Triggered"
```

Watchdog System Monitor Sample2 Policy

The second policy triggers an applet when the total amount of memory used by the process named Net Input is greater than 100 kb:

```
event manager applet IOSWD_Sample2
 event ioswdsysmon sub1 mem-proc taskname "Net Input" op gt val 100 is-percent false
 action 1.0 syslog msg "IOSWD_Sample2 Policy Triggered"
```

Watchdog System Monitor Sample3 Policy

The third policy triggers an applet when the total amount of memory used by the process named IP RIB Update has increased by more than 50 percent over the sample period of 60 seconds:

```
event manager applet IOSWD_Sample3
 event ioswdsysmon sub1 mem-proc taskname "IP RIB Update" op gt val 50 is-percent true
period 60
 action 1.0 syslog msg "IOSWD_Sample3 Policy Triggered"
```

The three policies are configured, and then repetitive large pings are made to the networking device from several workstations, causing the networking device to register some usage. This will trigger policies 1 and 2, and the console will display the following messages:

```
00:42:23: %HA_EM-6-LOG: IOSWD_Sample1: IOSWD_Sample1 Policy Triggered
00:42:47: %HA_EM-6-LOG: IOSWD_Sample2: IOSWD_Sample2 Policy Triggered
```

To view the policies that are registered, use the **show event manager policy registered** command:

```
Device# show event manager policy registered
No.  Class   Type    Event Type          Trap  Time Registered           Name
1    applet  system  ioswdsysmon         Off   Fri Jul 23 02:27:28 2004  IOSWD_Sample1
 sub1 cpu_util {taskname {IP Input} op ge val 1 period 10.000 }
 action 1.0 syslog msg "IOSWD_Sample1 Policy Triggered"
2    applet  system  ioswdsysmon         Off   Fri Jul 23 02:23:52 2004  IOSWD_Sample2
 sub1 mem_used {taskname {Net Input} op gt val 100 is_percent FALSE}
 action 1.0 syslog msg "IOSWD_Sample2 Policy Triggered"
3    applet  system  ioswdsysmon         Off   Fri Jul 23 03:07:38 2004  IOSWD_Sample3
 sub1 mem_used {taskname {IP RIB Update} op gt val 50 is_percent TRUE period 60.000 }
 action 1.0 syslog msg "IOSWD_Sample3 Policy Triggered"
```

Configuration SNMP Library Extensions Examples

SNMP Get Operations Examples

The following example shows how to send a get request to the local host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
 1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
90
Device(config-applet)# action 1.3 info type snmp oid
 1.3.6.1.2.1.1.1.0 get-type exact
community
 public
Device(config-applet)# action 1.3 info type snmp oid
 1.3.6.1.2.1.1.4.0 get-type next community
public
```

The following log message will be written to the SNMP event manager log:

```
1d03h:%HA_EM-6-LOG: lg: 1.3.6.1.2.1.1.1.0
1d04h:%HA_EM-6-LOG: lgn: 1.3.6.1.2.1.1.5.0
```

The following example shows how to send a get request to a remote host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
 1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
 90
Device(config-applet)# action 1.3 info type snmp oid
1.3.6.1.2.1.1.4.0 get-type next community
 public ipaddr
 172.17.16.69
Device(config-applet)# action 1.3 info type snmp getid
 1.3.6.1.2.1.1.1.0 community
 public ipaddr
 172.17.16.69
```

The following log message is written to the SNMP event manager log:

```
1d03h:%HA_EM-6-LOG: lg: 1.3.6.1.2.1.1.1.0
1d04h:%HA_EM-6-LOG: lgn: 1.3.6.1.2.1.1.5.0
```

SNMP GetID Operations Examples

The following example shows how to send a getid request to the local host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
5120000 poll-interval
 90
Device(config-applet)# action 1.3 info type snmp getid
 community
 public
```

The following log message is written to the SNMP event manager log:

```
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysname_oid=1.3.6.1.2.1.1.5.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysname_value=jubjub.cisco.com
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syslocation_oid=1.3.6.1.2.1.1.6.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syslocation_value=
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysdescr_oid=1.3.6.1.2.1.1.1.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysobjectid_oid=1.3.6.1.2.1.1.2.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysobjectid_value=products.222
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysuptime_oid=1.3.6.1.2.1.1.3.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysuptime_oid=10131676
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syscontact_oid=1.3.6.1.2.1.1.4.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syscontact_value=YYY
```

The following example shows how to send a getid request to a remote host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
 1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
 90
Device(config-applet)# action 1.3 info type snmp getid
 1.3.6.1.2.1.1.1.0 community
 public ipaddr
 172.17.16.69
```

The following log message is written to the SNMP event manager log:

```
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysname_oid=1.3.6.1.2.1.1.5.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysname_value=jubjub.cisco.com
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syslocation_oid=1.3.6.1.2.1.1.6.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syslocation_value=
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysdescr_oid=1.3.6.1.2.1.1.1.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysobjectid_oid=1.3.6.1.2.1.1.2.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysobjectid_value=products.222
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysuptime_oid=1.3.6.1.2.1.1.3.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_sysuptime_oid=10131676
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syscontact_oid=1.3.6.1.2.1.1.4.0
1d04h:%HA_EM-6-LOG: lgid: _info_snmp_syscontact_value=YYY
```

Set Operations Examples

The following example shows how to perform a set operation on the local host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
 1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
 90
Device(config-applet)# action 1.3 info type snmp oid
1.3.6.1.2.1.1.4.0 set-type
 integer
 5 sysName.0 community
 public
```

The following log message is written to the SNMP event manager log:

```
1d04h:%HA_EM-6-LOG: lset: 1.3.6.1.2.1.1.4.0
1d04h:%HA_EM-6-LOG: lset: XXX
```

The following example shows how to perform a set operation on a remote host.

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
1.3.6.1.2.1.1.1.0 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
90
Device(config-applet)# action 1.3 info type snmp oid
1.3.6.1.2.1.1.4.0 set-type integer
 5 sysName.0 community
 public ipaddr
172.17.16.69
```

The following log message is written to the SNMP event manager log:

```
1d04h:%HA_EM-6-LOG: lset: 1.3.6.1.2.1.1.4.0
1d04h:%HA_EM-6-LOG: lset: XXX
```

Generating SNMP Notifications Examples

The following example shows how to configure SNMP traps for the sysUpTime.0 variable:

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
 1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact entry-op
lt entry-val
 5120000 poll-interval
90
Device(config-applet)# action 1.3 info type snmp var
```

```
 sysUpTime.0 oid
1.3.6.1.4.1.9.9.43.1.1.6.1.3.41 integer
 2
Device(config-applet)# action 1.4 info type snmp trap
 enterprise-oid
 ciscoSyslogMIB.2 generic-trapnum
6 specific-trapnum
1 trap-oid
 1.3.6.1.4.1.9.9.41.2.0.1 trap-var
sysUpTime.0
```

The following output is generated if the debug snmp packets command is enabled:

```
Device# debug snmp packets
1d04h: SNMP: Queuing packet to 172.69.16.2
1d04h: SNMP: V1 Trap, ent ciscoSyslogMIB.2, addr 172.19.rap 1
clogHistoryEntry.3 = 4
clogHistoryEntry.6 = 9999
1d04h: SNMP: Queuing packet to 172.19.208.130
1d04h: SNMP: V1 Trap, ent ciscoSyslogMIB.2, addr 172.19.rap 1
clogHistoryEntry.3 = 4
clogHistoryEntry.6 = 9999
1d04h: SNMP: Packet sent via UDP to 172.69.16.2
1d04h: SNMP: Packet sent via UDP to 172.69.16.2
infra-view10:
Packet Dump:
30 53 02 01 00 04 04 63 6f 6d 6d a4 48 06 09 2b
06 01 04 01 09 09 29 02 40 04 ac 13 d1 17 02 01
06 02 01 01 43 04 00 9b 82 5d 30 29 30 12 06 0d
2b 06 01 04 01 09 09 29 01 02 03 01 03 02 01 04
30 13 06 0d 2b 06 01 04 01 09 09 29 01 02 03 01
06 02 02 27 0f
Received SNMPv1 Trap:
Community: comm
Enterprise: ciscoSyslogMIBNotificationPrefix
Agent-addr: 172.19.209.23
Enterprise Specific trap.
Enterprise Specific trap: 1
Time Ticks: 10191453
clogHistSeverity = error(4)
clogHistTimestamp = 9999
```

The following example shows how to configure SNMP inform requests for the sysUpTime.0 variable:

```
Device(config)# event manager applet snmp
Device(config-applet)# event snmp oid
1.3.6.1.4.1.9.9.48.1.1.1.6.1 get-type exact entry-op
 lt entry-val
 5120000 poll-interval
 90
Device(config-applet)# action 1.3 info type snmp var
 sysUpTime.0 oid
1.3.6.1.4.1.9.9.43.1.1.6.1.3.41 integer
 2
Device(config-applet)# action 1.4 info type snmp inform
trap-oid
 1.3.6.1.4.1.9.9.43.2.0.1 trap-var
 sysUpTime.0 community
 public ipaddr
 172.19.209.24
```

The following output is generated if the debug snmp packets command is enabled:

```
Device# debug snmp packets
1d04h: SNMP: Inform request, reqid 24, errstat 0, erridx 0
sysUpTime.0 = 10244391
snmpTrapOID.0 = ciscoConfigManMIB.2.0.1
ccmHistoryEventEntry.3.40 = 1
1d04h: SNMP: Packet sent via UDP to 172.19.209.24.162
1d04h: SNMP: Packet received via UDP from 172.19.209.24 on FastEthernet0/0
1d04h: SNMP: Response, reqid 24, errstat 0, erridx 0
1d04h: SNMP: Response, reqid 24, errstat 0, erridx 0
```

```
1d04h: SNMP: Inform request, reqid 25, errstat 0, erridx 0
sysUpTime.0 = 10244396
snmpTrapOID.0 = ciscoConfigManMIB.2.0.1
ccmHistoryEventEntry.3.41 = 2
1d04h: SNMP: Packet sent via UDP to 172.19.209.24.162
1d04h: SNMP: Packet received via UDP from 172.19.209.24 on FastEthernet0/0
1d04h: SNMP: Response, reqid 25, errstat 0, erridx 0
1d04h: SNMP: Response, reqid 25, errstat 0, erridx 0
Device# debug snmp packets
5d04h: SNMP: Packet received via UDP from 172.19.209.23 on FastEthernet0/0
5d04h: SNMP: Inform request, reqid 24, errstat 0, erridx 0
sysUpTime.0 = 10244391
snmpTrapOID.0 = ciscoConfigManMIB.2.0.1
ccmHistoryEventEntry.3.40 = 1
5d04h: dest if_index = 1
5d04h: dest ip addr= 172.19.209.24
5d04h: SNMP: Response, reqid 24, errstat 0, erridx 0
5d04h: SNMP: Packet sent via UDP to 172.19.209.23.57748
5d04h: SNMP: Packet received via UDP from 172.19.209.23 on FastEthernet0/0
5d04h: SNMP: Inform request, reqid 25, errstat 0, erridx 0
```

Configuring Variable Logic for EEM Applets Examples

The following sections provide examples on some selected action commands. For information on all the action commands supporting variable logic within applets, see the table below.

In this example, conditional loops **while**, **if** and **foreach** are used to print data. Other action commands such as **action divide**, **action increment** and **action puts** are used to define the actions to be performed when the conditions are met.

```
event manager applet printdata
event none
action 100 set colors "red green blue"
action 101 set shapes "square triangle rectange"
action 102 set i "1"
action 103 while $i lt 6
action 104  divide $i 2
action 105  if $_remainder eq 1
action 106   foreach _iterator "$colors"
action 107    puts nonewline "$_iterator "
action 108   end
action 109   puts ""
action 110  else
action 111   foreach _iterator "$shapes"
action 112    puts nonewline "$_iterator "
action 113   end
action 114   puts ""
action 115  end
action 116  increment i
action 117 end
```

When the event manager applet ex is run, the following output is obtained:

```
event manager run printdata
red green blue
square triangle rectange
red green blue
square triangle rectange
red green blue
```

In this example, two environment variables poll_interface and max_rx_rate are set to F0/0 and 3 respectively. Every 30 seconds there is a poll on an interface for rx rate. If the rx rate is greater than the threshold, a syslog message is displayed.

This applet makes use of the foreach conditional statement to poll the interface, the if conditional block to compare the value under RXPS with max_rx_rate that was set in the EEM environment variable.

```
event manager environment poll_interfaces F0/0
event manager environment max_rx_rate 3
ev man app check_rx_rate
ev timer watchdog name rx_timer time 30
action 100 foreach int $poll_interfaces
action 101  cli command "en"
action 102  cli command "show int $int summ | beg ------"
action 103  foreach line $_cli_result "\n"
action 105  regexp ".*[0-9]+\s+[0-9]+\s+[0-9]+\s+[0-9]+\s+[0-9]+\s+([0-9]+)\s+.*" $line
junk rxps
action 106   if $_regexp_result eq 1
action 107    if $rxps gt $max_rx_rate
action 108     syslog msg "Warning rx rate for $int is > than threshold. Current value is
$rxps
 (threshold is $max_rx_rate)"
action 109    end
action 110   end
action 111  end
action 112 end
```

Example syslog message:

```
Oct 16 09:29:26.153: %HA_EM-6-LOG: c: Warning rx rate for F0/0 is > than threshold.
 Current value is 4 (threshold is 3)
The output of show int F0/0 summ is of the format:

#show int f0/0 summ

 *: interface is up
 IHQ: pkts in input hold queue     IQD: pkts dropped from input queue
 OHQ: pkts in output hold queue    OQD: pkts dropped from output queue
 RXBS: rx rate (bits/sec)          RXPS: rx rate (pkts/sec)
 TXBS: tx rate (bits/sec)          TXPS: tx rate (pkts/sec)
 TRTL: throttle count

  Interface               IHQ   IQD  OHQ   OQD  RXBS RXPS  TXBS TXPS TRTL
------------------------------------------------------------------------------
* FastEthernet0/0          0 87283    0     0     0    0     0    0    0
```

Note To use other action commands supporting variable logic within applets, use the commands listed in the table below.

Table 10: Available action commands

Action Commands	Purpose
action add	Adds the value of two variables when an EEM applet is triggered.
action append	Appends the given value to the current value of a variable when an EEM applet is triggered.
action break	Causes an immediate exit from a loop of actions when an EEM applet is triggered
action comment	Adds comments to an applet when an EEM applet is triggered

Action Commands	Purpose
action context retrieve	Retrieves variables identified by a given set of context name keys when an EEM applet is triggered.
action context save	Saves information across multiple policy triggers when an EEM applet is triggered.
action continue	Continues with a loop of actions when an EEM applet is triggered.
action decrement	Decrements the value of a variable when an EEM applet is triggered.
action divide	Divides the dividend value by the given divisor value when an EEM applet is triggered.
action else	Specifies the beginning of else conditional action block in if / else conditional action block when an EEM applet is triggered.
action elseif	Identifies the beginning of the else conditional action block in the else / if conditional action block when an EEM applet is triggered.
action end	Specifies the identification of the end of an conditional action block in the if / else and while conditional action block when an EEM applet is triggered.
action exit	Specifies an immediate exit from the running applet configuration when an EEM applet is triggered.
action foreach	Specifies the iteration of an input string using the delimiter as a tokenizing pattern, when an EEM applet is triggered.
action gets	Gets an input from the local TTY in a synchronous applet and store the value in the given variable when an EEM applet is triggered.
action if	Specifies the identification of the beginning of an if conditional block when an EEM applet is triggered.
action if goto	Instructs the applet to jump to a given label if the specified condition is true when an EEM applet is triggered.
action increment	Increments the value of a variable when an EEM applet is triggered.
action info type interface-names	Specifies the action of obtaining interface names when an EEM applet is triggered.
action info type snmp getid	Retrieves the individual variables from a Simple Network Management Protocol (SNMP) entity during the SNMP get operation.
action info type snmp inform	Sends an SNMP inform requests when an EEM applet is triggered.

Action Commands	Purpose
action info type snmp oid	Specifies the type of SNMP get operation and the object to retrieve during the SNMP set operation, when an EEM applet is triggered.
action info type snmp trap	Sends SNMP trap requests when an EEM applet is triggered.
action info type snmp var	Creates a variable for an SNMP object identifier (OID) and its value from an EEM applet
action multiply	Specifies the action of multiplying the variable value with a specified given integer value when an EEM applet is triggered.
action puts	Enables the action of printing data directly to the local tty when an EEM applet is triggered.
action regexp	Specifies the action of matching a regular expression pattern on an input string when an EEM applet is triggered.
action set (EEM)	Specifies the action of setting the value of a variable when an EEM applet is triggered.
action string compare	Specifies the action of comparing two unequal strings when an EEM applet is triggered.
action string equal	Specifies the action of verifying whether or not two strings are equal when an EEM applet is triggered
action string first	Specifies the action of returning the index on the first occurrence of string1 within string2 when an EEM applet is triggered.
action string index	Specifies the action of returning the characters specified at a given index value when an EEM applet is triggered.
action string last	Specifies the action of returning the index on the last occurrence of string1 within string 2 when an EEM applet is triggered.
action string length	Specifies the action of returning the number of characters in a string when the EEM applet is triggered.
action string match	Specifies the action of returning 1 to the $_string_result, if the string matches the pattern when an EEM applet is triggered.
action string range	Specifies the action of storing a range of characters in a string when an EEM applet is triggered.
action string replace	Specifies the action of storing a new string by replacing range of characters in the specified string when an EEM applet is triggered.
action string tolower	Specifies the action of storing specific range of characters of a string in lowercase when an EEM applet is triggered.

Action Commands	Purpose
action string toupper	Specifies the action of storing specific range of characters of a string in uppercase when an EEM applet is triggered.
action string trim	Specifies the action to trim a string when an EEM applet is triggered.
action string trimleft	Specifies the action to trim the characters of one string from the left end of another string when an EEM applet is triggered.
action string trimright	Specifies the action to trim the characters one string from the right end of another string when an EEM applet is triggered.
action subtract	Subtracts the value of a variable from another value when an EEM applet is triggered.
action while	Specifies the action of identifying the beginning of a loop of conditional block when an EEM applet is triggered.

Configuring Event SNMP-Object Examples

The following example shows the SET operation and the value to set is in $_snmp_value and it is managed by the script. The example below saves the oid and its value as contexts to be retrieved later.

```
event manager applet snmp-object1
 description "APPLET SNMP-OBJ-1"
 event snmp-object oid 1.3.6.1.2.1.31.1.1.1.18 type string sync no skip no istable yes
default 0
 action 1 syslog msg "SNMP-OBJ1:TRIGGERED" facility "SNMP_OBJ"
 action 2 context save key myoid variable "_snmp_oid"
 action 3 context save key myvalue variable "_snmp_value"
```

Configuring Description of an EEM Applet Examples

The following example shows how to add or modify the description for an Embedded Event Manager (EEM) applet that is run by sampling Simple Network Management Protocol (SNMP):

```
event manager applet test
 description "This applet looks for the word count in syslog messages"
 event syslog pattern "count"
 action 1 syslog msg hi
```

Additional References

The following sections provide references related to writing EEM policies Using the Cisco IOS CLI.

Related Documents

Related Topic	Document Title
Cisco IOS commands	Cisco IOS Master Commands List, All Releases
EEM commands: complete command syntax, defaults, command mode, command history, usage guidelines, and examples	Cisco IOS Embedded Event Manager Command Reference
Embedded Event Manager overview	Embedded Event Manager Overview module
Embedded Event Manager policy writing using Tcl	Writing Embedded Event Manager Policies Using Tcl module
Configuring enhanced object tracking	Configuring Enhanced Object Tracking module

Standards

Standard	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--

MIBs

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

RFCs

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

Technical Assistance

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

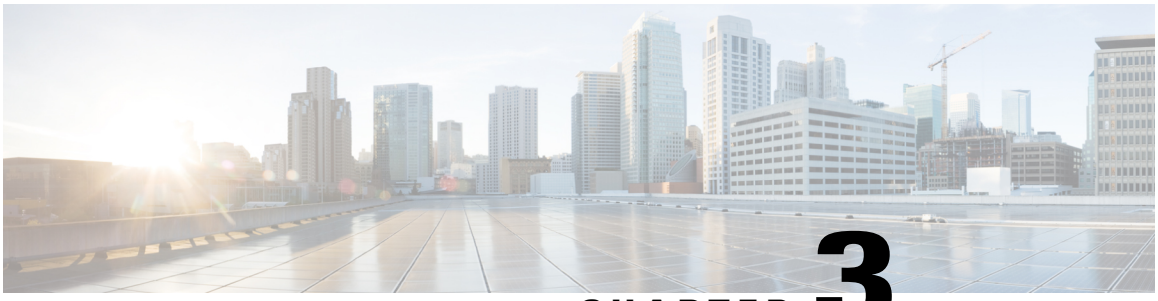
Feature Information for Writing EEM 3.2 Policies Using the Cisco IOS CLI

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

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

Table 11: Feature Information for Writing EEM 3.2 Policies Using the Cisco IOS CLI

Feature Name	Releases	Feature Information
Embedded Event Manager 3.2	Cisco IOS XE Release 3.3SE	The following sections provide information about this feature: In Cisco IOS XE Release 3.3SE, this feature is supported on Cisco Catalyst 3850 Series Switches and Cisco 5700 Wireless LAN Controllers. The following commands were introduced or modified: **debug event manager**, **event identity**, **event mat**, **event neighbor-discovery**, **show event manager detector**.



CHAPTER 3

Writing Embedded Event Manager Policies Using Tcl

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

- Finding Feature Information, page 107
- Prerequisites for Writing Embedded Event Manager Policies Using Tcl, page 108
- Information About Writing Embedded Event Manager Policies Using Tcl, page 108
- How to Write Embedded Event Manager Policies Using Tcl, page 115
- Configuration Examples for Writing Embedded Event Manager Policies Using Tcl, page 146
- Additional References, page 167
- Feature Information for Writing Embedded Event Manager 3.2 Policies Using Tcl, page 168

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest caveats and feature information, see Bug Search Tool and the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the feature information table.

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

Prerequisites for Writing Embedded Event Manager Policies Using Tcl

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

Information About Writing Embedded Event Manager Policies Using Tcl

EEM Policies

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

EEM Applet

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

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

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

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

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

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

EEM Script

All Embedded Event Manager scripts are written in Tcl. Tcl is a string-based command language that is interpreted at run time. The version of Tcl supported is Tcl version 8.3.4 plus added script support. Scripts are defined using an ASCII editor on another device, not on the networking device. The script is then copied to the networking device and registered with EEM. Tcl scripts are supported by EEM. As an enforced rule, Embedded Event Manager policies are short-lived run time routines that must be interpreted and executed in less than 20 seconds of elapsed time. If more than 20 seconds of elapsed time are required, the maxrun parameter may be specified in the event_register statement to specify any desired value.

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

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

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

EEM Policy Tcl Command Extension Categories

There are different categories of EEM policy Tcl command extensions.

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

Table 12: EEM Policy Tcl Command Extension Categories

Category	Definition
EEM event Tcl command extensions (three types: event information, event registration, and event publish)	This category is represented by the **event_register_** *xxx* family of event-specific commands. There is a separate event information Tcl command extension in this category as well: **event_reqinfo**. This is the command used in policies to query the EEM for information about an event. There is also an EEM event publish Tcl command extension **event_publish**> *that publishes an application-specific event.*

Category	Definition
EEM action Tcl command extensions	These Tcl command extensions (for example, **action_syslog**) are used by policies to respond to or recover from an event or fault. In addition to these extensions, developers can use the Tcl language to implement any action desired.
EEM utility Tcl command extensions	These Tcl command extensions are used to retrieve, save, set, or modify application information, counters, or timers.
EEM system information Tcl command extensions	This category is represented by the **sys_reqinfo _** *xxx* family of system-specific information commands. These commands are used by a policy to gather system information.
EEM context Tcl command extensions	These Tcl command extensions are used to store and retrieve a Tcl context (the visible variables and their values).

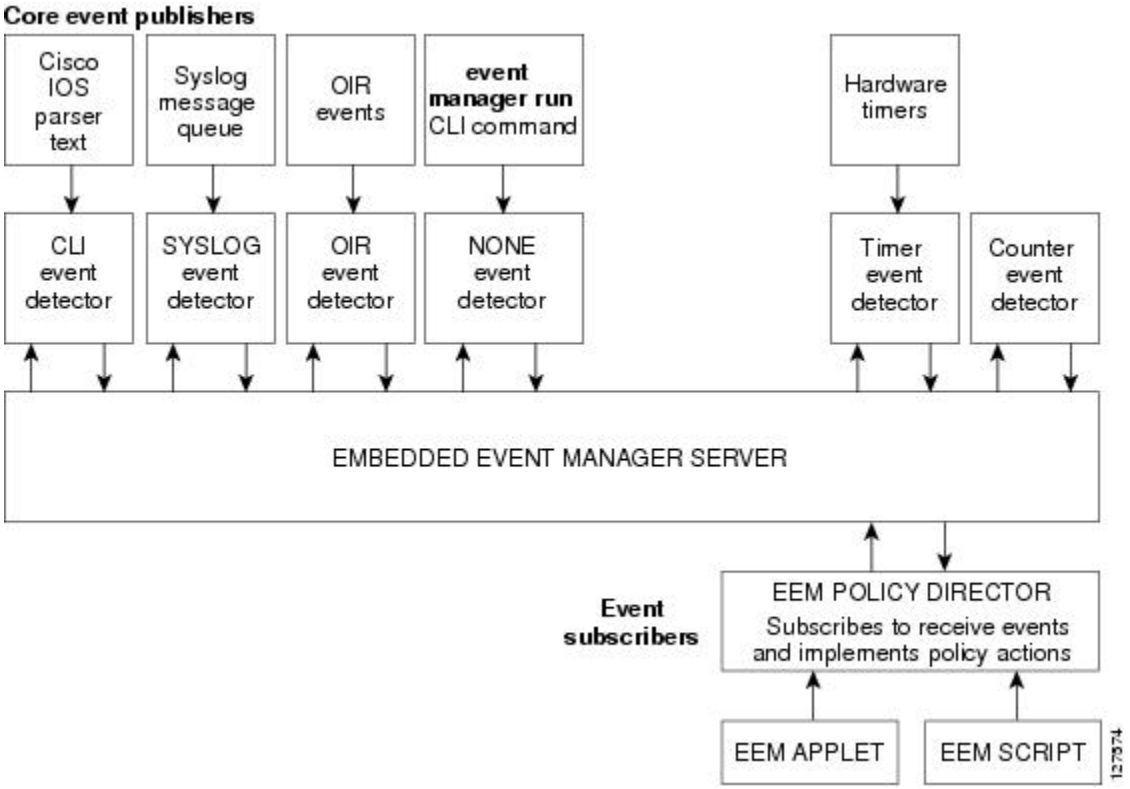
General Flow of EEM Event Detection and Recovery

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

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

Tcl command extension **event_reqinfo** is used in the policy to query the Embedded Event Manager for information about the detected event.

Figure 3: Embedded Event Manager Core Event Detectors



Safe-Tcl

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

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

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

- **cd** --Change directory is not allowed to one of the restricted Cisco directory names.
- **encoding** --The commands **encoding names**, **encoding convertfrom**, and **encoding convertto** are permitted. The **encoding system** command with no arguments is permitted, but the **encoding system** command with the **?encoding?** keyword is not permitted.
- **exec** --Not permitted.

- **fconfigure** --Permitted.
- **file** --The following are permitted:
  - **file dirname**
  - **file exists**
  - **file extension**
  - **file isdirectory**
  - **file join**
  - **file pathtype**
  - **file rootname**
  - **file split**
  - **file stat**
  - **file tail**
- **file** --The following are not permitted:
  - **file atime**
  - **file attributes**
  - **file channels**
  - **file copy**
  - **file delete**
  - **file executable**
  - **file isfile**
  - **file link**
  - **file lstat**
  - **file mkdir**
  - **file mtime**
  - **file nativename**
  - **file normalize**
  - **file owned**
  - **file readable**
  - **file readlink**
  - **file rename**
  - **file rootname**
  - **file separator**
  - **file size**

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

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

Bytecode Support for EEM 2.4

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

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

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

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

Registration Substitution

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

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

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

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

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

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

```
$_eem_syslog_statement occurs $_eem_occurs_val
action_syslog “this is test 3”
```

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

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

Cisco File Naming Convention for EEM

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

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

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

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

Table 13: Two-Character Abbreviation Specification

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

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

How to Write Embedded Event Manager Policies Using Tcl

Registering and Defining an EEM Tcl Script

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

Before You Begin

You must have a policy available that is written in the Tcl scripting language. Sample policies are provided--see the details in the Sample EEM Policies, on page 126 to see which policies are available for the Cisco IOS release image that you are using--and these sample policies are stored in the system policy directory.

SUMMARY STEPS

1. **enable**
2. **show event manager environment** [**all**| *variable-name*]
3. **configure terminal**
4. **event manager environment** *variable-name string*
5. Repeat Registering and Defining an EEM Tcl Script to configure all the environment variables required by the policy to be registered in Registering and Defining an EEM Tcl Script.
6. **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]
7. **exit**

DETAILED STEPS

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

	Command or Action	Purpose
	Example: `Device(config)# event manager policy tm_cli_cmd.tcl type system`	• Use the **system** keyword to register a Cisco-defined system policy. • Use the **user** keyword to register a user-defined system policy. • Use the **trap** keyword to generate an SNMP trap when the policy is triggered. • In this example, the sample EEM policy named tm_cli_cmd.tcl is registered as a system policy.
Step 7	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

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

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

Displaying EEM Registered Policies

Perform this optional task to display EEM registered policies.

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

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

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

Example:

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

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

Example:

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

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

Example:

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

Unregistering EEM Policies

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

SUMMARY STEPS

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

DETAILED STEPS

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

	Command or Action	Purpose
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.
Step 6	Repeat Unregistering EEM Policies to ensure that the policy has been removed. **Example:** `Device# show event manager policy registered`	--

Examples

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

```
Device# show event manager policy registered
No.  Type    Event Type          Trap  Time Registered           Name
1    system  timer cron          Off   Tue Oct11  01:43:18 2005 tm_cli_cmd.tcl
 name {crontimer2} cron entry {0-59/1 0-23/1 * * 0-7}
 nice 0 priority normal maxrun 240.000
2    system  syslog              Off   Tue Oct11  01:43:28 2005 sl_intf_down.tcl
 occurs 1 pattern {.*UPDOWN.*Ethernet1/0.*}
 nice 0 priority normal maxrun 90.000
3    system  proc abort          Off   Tue Oct11  01:43:38 2005 pr_cdp_abort.tcl
 instance 1 path {cdp2.iosproc}
 nice 0 priority normal maxrun 20.000
```

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

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

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

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

Suspending EEM Policy Execution

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

SUMMARY STEPS

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

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager policy registered** [**event-type** *event-name*][**system**| **user**] [**time-ordered**| **name-ordered**] [**detailed** *policy-filename*] **Example:** `Device# show event manager policy registered`	(Optional) Displays the EEM policies that are currently registered. • The optional **system**or **user** keyword displays the registered system or user policies. • If no keywords are specified, EEM registered policies for all event types are displayed in time order.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**event manager scheduler suspend** **Example:** `Device(config)# event manager scheduler suspend`	Immediately suspends the execution of all EEM policies.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

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

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

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

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

Managing EEM Policies

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

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

SUMMARY STEPS

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

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show event manager directory user** [**library**| **policy**]	(Optional) Displays the directory to use for storing EEM user library or policy files.

	Command or Action	Purpose
	Example: `Device# show event manager directory user library`	• The optional **library**keyword displays the directory to use for user library files. • The optional **policy**keyword displays the directory to use for user-defined EEM policies.
Step 3	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 4	**event manager directory user** {**library** *path*| **policy** *path*} **Example:** `Device(config)# event manager directory user library disk0:/user_library`	Specifies a directory to use for storing user library files or user-defined EEM policies. • Use the *path*argument to specify the absolute pathname to the user directory.
Step 5	**exit** **Example:** `Device(config)# exit`	Exits global configuration mode and returns to privileged EXEC mode.

Examples

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

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

Modifying History Table Size and Displaying EEM History Data

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

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **configure terminal**
Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 3 **event manager history size** {**events** | **traps**} [*size*]
Use this command to change the size of the EEM event history table or the size of the EEM SNMP trap history table. In the following example, the size of the EEM event history table is changed to 30 entries:

Example:

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

Step 4 **exit**
Exits global configuration mode and returns to privileged EXEC mode.

Example:

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

Step 5 **show event manager history events** [**detailed**] [**maximum** *number*]
Use this command to display information about each EEM event that has been triggered.

Example:

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

Step 6 **show event manager history traps** [**server** | **policy**]
Use this command to display the EEM SNMP traps that have been sent either from the EEM server or from an EEM policy.

Example:

```
Device# show event manager history traps
No.  Time                     Trap Type           Name
```

```
1    Fri Sep  9 13:48:40 2005  server              applet: four
2    Fri Sep  9 13:57:03 2005  policy              script: no_snmp_test.tcl
```

Displaying Software Modularity Process Reliability Metrics Using EEM

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

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **show event manager metric process** {**all**| *process-name*}
Use this command to display the reliability metric data for processes. The system keeps a record of when processes start and end, and this data is used as the basis for reliability analysis. In this partial example, the first and last entries showing the metric data for the processes on all the cards inserted in the system are displayed.

Example:

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

```
.
.
.
====================================
process name: cdp2.iosproc, instance: 1
sub_system id: 0, version: 00.00.0000
--------------------------------
last event type: process start
recent start time: Fri Oct10  20:35:02 2005
recent normal end time: n/a
recent abnormal end time: n/a
number of times started: 1
number of times ended normally: 0
number of times ended abnormally: 0
most recent 10 process start times:
--------------------------
Fri Oct10  20:35:02 2005
--------------------------
most recent 10 process end times and types:

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

Troubleshooting Tips

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

Modifying the Sample EEM Policies

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

Sample EEM Policies

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

The table below describes the sample EEM policies.

Table 14: Sample EEM Policy Descriptions

Name of Policy	Description
pr_cdp_abort.tcl	Introduced with Cisco Software Modularity images. This policy monitors for cdp2.iosproc process abort events. It will log a message to SYSLOG and send an e-mail with the details of the abort.
pr_crash_reporter.tcl	Introduced with Cisco Software Modularity images. This policy monitors for all process abort events. When an event occurs, the policy will send crash information, including the crashdump file, to the specified URL where a CGI script processes the data.
pr_iprouting_abort.tcl	Introduced with Cisco Software Modularity images. This policy monitors for iprouting.iosproc process abort events. It will log a message to SYSLOG and send an e-mail with the details of the abort.
sl_intf_down.tcl	This policy runs when a configurable syslog message is logged. It will execute a configurable CLI command and e-mail the results.
tm_cli_cmd.tcl	This policy runs using a configurable CRON entry. It will execute a configurable CLI command and e-mail the results.
tm_crash_history.tcl	Introduced with Cisco Software Modularity images. This policy runs at midnight every day and e-mails a process crash history report to a specified e-mail address.
tm_crash_reporter.tcl	This policy runs 5 seconds after it is registered. If the policy is saved in the configuration, it will also run each time that the device is reloaded. The policy will prompt for the reload reason. If the reload was due to a crash, the policy will search for the latest crashinfo file and send this information to a specified URL location.
tm_fsys_usage.tcl	Introduced with Cisco Software Modularity images. This policy runs using a configurable CRON entry and monitors disk space usage. A syslog message will be displayed if disk space usage crosses configurable thresholds.

Name of Policy	Description
wd_mem_reporter.tcl	Introduced with Cisco Software Modularity images. This policy reports on low system memory conditions when the amount of memory available falls below 20 percent of the initial available system memory. A syslog message will be displayed and, optionally, an e-mail will be sent.

For more details about the sample policies available and how to run them, see the EEM Event Detector Demo Examples, on page 147.

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **show event manager policy available detailed** *policy-filename*
Displays the actual specified sample policy including details about the environment variables used by the policy and instructions for running the policy. The **detailed** keyword was introduced for the **show event manager policy available** and the **show event manager policy registered** commands. Depending on your release, you may need to copy one of the two Tcl scripts from the configuration examples section in this document (see the Programming Policies with Tcl Sample Scripts Example). In the following example, details about the sample policy tm_cli_cmd.tcl are displayed on the screen.

Example:

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

Step 3 Cut and paste the contents of the sample policy displayed on the screen to a text editor.
Use the edit and copy functions to move the contents from the device to a text editor on another device.

Step 4 Edit the policy and save it with a new filename.
Use the text editor to modify the policy as a Tcl script. For file naming conventions, see the Cisco File Naming Convention for EEM, on page 114.

Step 5 Copy the new file back to the device flash memory.
Copy the file to the flash file system on the device--typically disk0:. For more details about copying files, see the "Using the Cisco IOS File System" chapter in the *Configuration Fundamentals Configuration Guide*.

Step 6 **configure terminal**
Enters global configuration mode.

Example:

```
Device# configure terminal
```

Step 7 **event manager directory user** {**library** *path*| **policy** *path*}
Specifies a directory to use for storing user library files or user-defined EEM policies. In the following example, the user_library directory on disk0 is specified as the directory for storing user library files.

Example:

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

Step 8 **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]
Registers the EEM policy to be run when the specified event defined within the policy occurs. In the following example, the new EEM policy named test.tcl is registered as a user-defined policy.

Example:

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

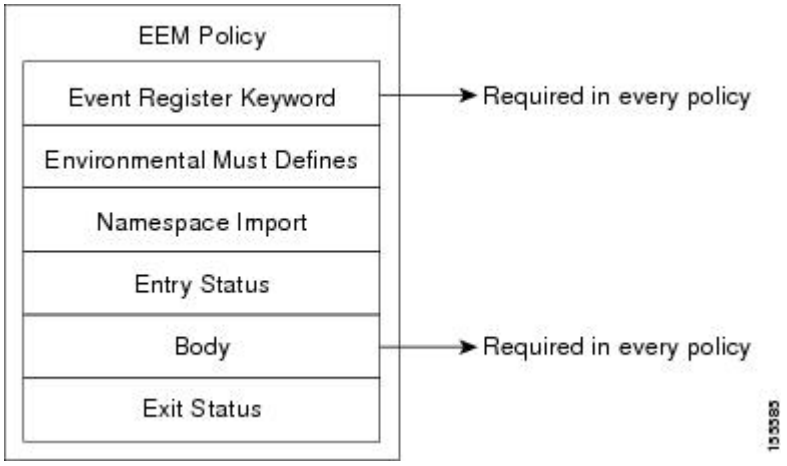
Programming EEM Policies with Tcl

Perform this task to help you program a policy using Tcl command extensions. We recommend that you copy an existing policy and modify it. There are two required parts that must exist in an EEM Tcl policy: the **event_register** Tcl command extension and the body. All other sections shown in the Tcl Policy Structure and Requirements, on page 130 concept are optional.

Tcl Policy Structure and Requirements

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

Figure 4: Tcl Policy Structure and Requirements



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

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

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

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

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

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

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

- The **event_reqinfo**event information Tcl command extension that is used to query the EEM for information about the detected event.

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

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

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

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

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

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

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

EEM Entry Status

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

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

EEM Exit Status

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

EEM Policies and Cisco Error Number

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

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

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

_cerrno: 32-Bit Error Return Values

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

```
XYSSSSSSSSSSSSSEEEEEEEEPPPPPPPPP
```

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

```
862439AE
```

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

```
10000110001001000011100110101110
```

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

Table 15: _cerrno: 32-Bit Error Return Value Variables

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

Variable	Description
PPPPPPPP	The pass-through POSIX error code (9 bits = 512 values). This represents the last of the 32-bit sequence, and the string corresponding to this error code is contained in $_cerr_posix_err.

Error Class Encodings for XY

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

Table 16: Error Class Encodings

00	CERR_CLASS_SUCCESS
01	CERR_CLASS_INFO
10	CERR_CLASS_WARNING
11	CERR_CLASS_FATAL

An error return value of zero means SUCCESS.

SUMMARY STEPS

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

DETAILED STEPS

Step 1 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 **show event manager policy available detailed** *policy-filename*
Displays the actual specified sample policy including details about the environment variables used by the policy and instructions for running the policy. The **detailed** keyword was introduced for the **show event manager policy available** and the **show event manager policy registered** commands. Depending on your release, you must copy one of the two Tcl scripts from the configuration examples section in this document (see the Programming Policies with Tcl Sample Scripts Example). In the following example, details about the sample policy tm_cli_cmd.tcl are displayed on the screen.

Example:

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

Step 3 Cut and paste the contents of the sample policy displayed on the screen to a text editor.
Use the edit and copy functions to move the contents from the device to a text editor on another device. Use the text editor to edit the policy as a Tcl script.

Step 4 Define the required **event_register** Tcl command extension.
Choose the appropriate **event_register** Tcl command extension from the table below for the event that you want to detect, and add it to the policy.

Table 17: EEM Event Registration Tcl Command Extensions

Event Registration Tcl Command Extensions
event_register_appl
event_register_cli
event_register_counter
event_register_gold
event_register_interface
event_register_ioswdsysmon
event_register_ipsla
event_register_nf
event_register_none

Event Registration Tcl Command Extensions
event_register_oir
event_register_process
event_register_resource
event_register_rf
event_register_routing
event_register_rpc
event_register_snmp
event_register_snmp_notification
event_register_snmp_object
event_register_syslog
event_register_timer
event_register_timer_subscriber
event_register_track
event_register_wdsysmon

Step 5 Add the appropriate namespace under the ::cisco hierarchy.
Policy developers can use the new namespace ::cisco in Tcl policies in order to group all the extensions used by Cisco IOS EEM. There are two namespaces under the ::cisco hierarchy, and the table below shows which category of EEM Tcl command extension belongs under each namespace.

Table 18: Cisco IOS EEM Namespace Groupings

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

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

Step 6 Program the must defines section to check for each environment variable that is used in this policy.
This is an optional step. Must defines are a section of the policy that tests whether any EEM environment variables that are required by the policy are defined before the recovery actions are taken. The must defines section is not required if the policy does not use any EEM environment variables. EEM environment variables for EEM scripts are Tcl global variables that are defined external to the policy before the policy is run. To define an EEM environment variable, use the Embedded Event Manager configuration command **event manager environment** CLI command. By convention all Cisco EEM environment variables begin with " _ " (an underscore). In order to avoid future conflict, customers are urged not to define new variables that start with " _ ".

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

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

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

Table 19: E-mail-Specific Environmental Variables Used by the Sample Policies

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

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

Example of Must Defines

Example:

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

Step 7 Program the body of the script.
In this section of the script, you can define any of the following:

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

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

Step 8 Check the entry status to determine if a policy has previously run for this event.
If the prior policy is successful, the current policy may or may not require execution. Entry status designations may use one of three possible values: 0 (previous policy was successful), Not=0 (previous policy failed), and Undefined (no previous policy was executed).

Step 9 Check the exit status to determine whether or not to apply the default action for this event, if a default action exists.
A value of zero means do not perform the default action. A value of nonzero means perform the default action. The exit status will be passed to subsequent policies that are run for the same event.

Step 10 Set Cisco Error Number (_cerrno) Tcl global variables.
Some EEM Tcl command extensions set a Cisco Error Number Tcl global variable _cerrno. Whenever _cerrno is set, four other Tcl global variables are derived from _cerrno and are set along with it (_cerr_sub_num, _cerr_sub_err, _cerr_posix_err, and _cerr_str).

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

Example:

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

Step 11 Save the Tcl script with a new filename, and copy the Tcl script to the device.
Embedded Event Manager policy filenames adhere to the following specification:

- An optional prefix--Mandatory.--indicating, if present, that this is a system policy that should be registered automatically at boot time if it is not already registered. For example: Mandatory.sl_text.tcl.
- A filename body part containing a two-character abbreviation (see EEM Policies and Cisco Error Number, on page 132) for the first event specified; an underscore character part; and a descriptive field part further identifying the policy.
- A filename suffix part defined as .tcl.

For more details, see the Cisco File Naming Convention for EEM, on page 114.

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

Step 12 **configure terminal**
Enters global configuration mode.

Example:

```
Device# configure terminal
```

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

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

Example:

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

Step 14 **event manager policy** *policy-filename* [**type** {**system**| **user**}] [**trap**]
Registers the EEM policy to be run when the specified event defined within the policy occurs. In the following example, the new EEM policy named cl_mytest.tcl is registered as a user-defined policy.

Example:

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

Step 15 Cause the policy to execute, and observe the policy.
To test that the policy runs, generate the conditions that will cause the policy to execute and observe that the policy runs as expected.

Step 16 Use debugging techniques if the policy does not execute correctly.
Use the Cisco IOS **debug event manager** CLI command with its various keywords to debug issues. Refer to the Troubleshooting Tips, on page 139 for details about using Tcl-specific keywords.

Troubleshooting Tips

- Use the **debug event manager tcl commands** CLI command to debug issues with Tcl extension commands. When enabled, this command displays all data that is passed in and read back from the TTY session that handles the CLI interactions. This data helps ensure users that the commands they are passing to the CLI are valid.
- The CLI library allows users to run CLI commands and obtain the output of commands in Tcl. Use the **debug event manager tcl cli-library** CLI command to debug issues with the CLI library.
- The SMTP library allows users to send e-mail messages to an SMTP e-mail server. Use the **debug event manager tcl smtp_library** CLI command to debug issues with the SMTP library. When enabled, this command displays all data that is passed in and read back from the SMTP library routines. This data helps ensure users that the commands they are passing to the SMTP library are valid.
- Tcl is a flexible language that allows you to override commands. For example, you can modify the **set** command and create a version of the **set** command that displays a message when a scalar variable is set. When the **set** command is entered in a policy, a message is displayed anytime a scalar variable is set, and this provides a way to debug scalar variables. To view an example of this debugging technique, see the Tracing Tcl set Command Operations Example, on page 165.

To view examples of the some of these debugging techniques, see the Debugging Embedded Event Manager Policies Examples, on page 163.

Creating an EEM User Tcl Library Index

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

SUMMARY STEPS

1. On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl library files into the directory.
2. **tclsh**
3. **auto_mkindex** *directory_name* ***.tcl**
4. Copy the Tcl library files from Creating an EEM User Tcl Library Index, on page 140 and the tclIndex file from Creating an EEM User Tcl Library Index, on page 140 to the directory used for storing user library files on the target device.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device. The directory can be the same directory used in Creating an EEM User Tcl Library Index, on page 140.
6. **enable**
7. **configure terminal**
8. **event manager directory user library** *path*
9. **event manager directory user policy** *path*
10. **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
11. **event manager run** *policy-name*

DETAILED STEPS

Step 1 On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl library files into the directory. The following example files can be used to create a tclIndex on a workstation running the Tcl shell:

lib1.tcl

Example:

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

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

lib2.tcl

Example:

```
proc test3 {} {
```

```
    puts "In procedure test3"
}
```

Step 2 **tclsh**

Use this command to enter the Tcl shell.

Example:

```
workstation% tclsh
```

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

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

Example:

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

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

tclIndex

Example:

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

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

Step 4 Copy the Tcl library files from Creating an EEM User Tcl Library Index, on page 140 and the tclIndex file from Creating an EEM User Tcl Library Index, on page 140 to the directory used for storing user library files on the target device.

Step 5 Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device. The directory can be the same directory used in Creating an EEM User Tcl Library Index, on page 140. The directory for storing user-defined EEM policies can be the same directory used in Creating an EEM User Tcl Library Index, on page 140. The following example user-defined EEM policy can be used to test the Tcl library support in EEM.

libtest.tcl

Example:

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

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

global auto_index auto_path

puts [array names auto_index]
```

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

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

Step 6 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 7 **configure terminal**
Enables global configuration mode.

Example:

```
Device# configure terminal
```

Step 8 **event manager directory user library** *path*
Use this command to specify the EEM user library directory; this is the directory to which the files in Creating an EEM User Tcl Library Index, on page 140 were copied.

Example:

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

Step 9 **event manager directory user policy** *path*
Use this command to specify the EEM user policy directory; this is the directory to which the file in Creating an EEM User Tcl Library Index, on page 140 was copied.

Example:

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

Step 10 **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
Use this command to register a user-defined EEM policy. In this example, the policy named libtest.tcl is registered.

Example:

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

Step 11 **event manager run** *policy-name*
Use this command to manually run an EEM policy. In this example, the policy named libtest.tcl is run to test the Tcl support in EEM. The example output shows that the test for Tcl support in EEM was successful.

Example:

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

Creating an EEM User Tcl Package Index

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

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

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

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

SUMMARY STEPS

1. On your workstation (UNIX, Linux, PC, or Mac) create a library directory and copy the Tcl package files into the directory.
2. **tclsh**
3. **pkg_mkindex** *directory_name* ***.tcl**
4. Copy the Tcl library files from Creating an EEM User Tcl Package Index, on page 143 and the pkgIndex file from Creating an EEM User Tcl Package Index, on page 143 to the directory used for storing user library files on the target device.
5. Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device. The directory can be the same directory used in Creating an EEM User Tcl Package Index, on page 143.
6. **enable**
7. **configure terminal**
8. **event manager directory user library** *path*
9. **event manager directory user policy** *path*
10. **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
11. **event manager run** *policy-name*

DETAILED STEPS

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

Step 2 **tclsh**

Use this command to enter the Tcl shell.

Example:

```
workstation% tclsh
```

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

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

Example:

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

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

pkgIndex

Example:

```
# Tcl package index file, version 1.1
```

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

Step 4 Copy the Tcl library files from Creating an EEM User Tcl Package Index, on page 143 and the pkgIndex file from Creating an EEM User Tcl Package Index, on page 143 to the directory used for storing user library files on the target device.

Step 5 Copy a user-defined EEM policy file written in Tcl to the directory used for storing user-defined EEM policies on the target device. The directory can be the same directory used in Creating an EEM User Tcl Package Index, on page 143. The directory for storing user-defined EEM policies can be the same directory used in Creating an EEM User Tcl Package Index, on page 143. The following example user-defined EEM policy can be used to test the Tcl package support in EEM.

packagetest.tcl

Example:

```
::cisco::eem::event_register_none maxrun 1000000.000
#
# test if xmlrpc available
#
#
# Namespace imports
#
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
#
package require xmlrpc
puts "Did you get an error?"
```

Step 6 **enable**
Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 7 **configure terminal**
Enables global configuration mode.

Example:

```
Device# configure terminal
```

Step 8 **event manager directory user library** *path*
Use this command to specify the EEM user library directory; this is the directory to which the files in Creating an EEM User Tcl Package Index, on page 143 were copied.

Example:

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

Step 9 **event manager directory user policy** *path*
Use this command to specify the EEM user policy directory; this is the directory to which the file in Creating an EEM User Tcl Package Index, on page 143 was copied.

Example:

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

Step 10 **event manager policy** *policy-name* [**type** {**system** | **user**} [**trap**]
Use this command to register a user-defined EEM policy. In this example, the policy named packagetest.tcl is registered.

Example:

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

Step 11 **event manager run** *policy-name*
Use this command to manually run an EEM policy. In this example, the policy named packagetest.tcl is run to test the Tcl package support in EEM.

Example:

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

Configuration Examples for Writing Embedded Event Manager Policies Using Tcl

Assigning a Username for a Tcl Session Examples

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

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

EEM Event Detector Demo Examples

EEM Sample Policy Descriptions

This configuration example features some of the sample EEM policies:

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

Event Manager Environment Variables for the Sample Policies

Event manager environment variables are Tcl global variables that are defined external to the EEM policy before the policy is registered and run. The sample policies require three of the e-mail environment variables to be set (see EEM Event Detector Demo Examples, on page 147 for a list of the e-mail variables); only _email_cc is optional. Other required and optional variable settings are outlined in the following tables.

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

Table 20: Environment Variables Used in the ap_perf_test_base_cpu.tcl Policy

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

Environment Variable	Description	Example
_perf_cmd3	The third non interactive CLI command that is as part of the measurement test. To use _perf_cmd3, _perf_cmd1 must be defined. This variable is optional and need not be specified.	**show interface counters protocol status**

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

Table 21: Environment Variables Used in the no_perf_test_init.tcl Policy

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

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

Table 22: Environment Variables Used in the sl_intf_down.tcl Policy

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

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

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

Table 23: Environment Variables Used in the tm_cli_cmd.tcl Policy

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

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

Table 24: Environment Variables Used in the tm_crash_reporter.tcl Policy

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

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

Table 25: Environment Variables Used in the tm_fsys_usage.tcl Policy

Environment Variable	Description	Example
_tm_fsys_usage_cron	A CRON specification that is used in the **event_register** Tcl command extension. If unspecified, the tm_fsys_usage.tcl policy is triggered once per minute. This variable is optional and need not be specified.	0-59/1 0-23/1 * * 0-7
_tm_fsys_usage_debug	When this variable is set to a value of 1, disk usage information is displayed for all entries in the system. This variable is optional and need not be specified.	1
_tm_fsys_usage_ freebytes	Free byte threshold for systems or specific prefixes. If free space falls below a given value, a warning is displayed. This variable is optional and need not be specified.	disk2:98000000
_tm_fsys_usage_percent	Disk usage percentage thresholds for systems or specific prefixes. If the disk usage percentage exceeds a given percentage, a warning is displayed. If unspecified, the default disk usage percentage is 80 percent for all systems. This variable is optional and need not be specified.	nvram:25 disk2:5

Registration of Some EEM Policies

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

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

Basic Configuration Details for All Sample Policies

To allow e-mail to be sent from the Embedded Event Manager, the **hostname** and **ip domain-name** commands must be configured. The EEM environment variables must also be set. After a Cisco IOS image has been

booted, use the following initial configuration, substituting appropriate values for your network. The environment variables for the tm_fsys_usage sample policy (see the table above) are all optional and are not listed here:

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

Using the Sample Policies

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

Running the Mandatory.go_*.tcl Sample Policy

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

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

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

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

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

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

Running the ap_perf_test_base_cpu.tcl and no_perf_test_init.tcl Sample Policies

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

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

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

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

Running the no_perf_test_init.tcl Sample Policy

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

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

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

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

Running the sl_intf_down.tcl Sample Policy

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

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

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

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

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

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

Running the tm_cli_cmd.tcl Sample Policy

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

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

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

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

Running the tm_crash_reporter.tcl Sample Policy

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

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

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

can register the tm_crash_reporter.tcl policy with EEM using the **event manager policy** command. Exit from global configuration mode and enter the **show event manager policy registered** command to verify that the policy has been registered.

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

Running the tm_fsys_usage.tcl Sample Policy

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

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

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

Programming Policies with Tcl Sample Scripts Example

This section contains some of the sample policies that are included as EEM system policies. For more details about these policies, see the EEM Event Detector Demo Examples, on page 147.

Mandatory.go_ipsec.tcl Sample Policy

The following sample policy for the TestIPSecEncrypDecrypPkt Test.

```
::cisco::eem::event_register_gold card all testing_type monitoring test_name Tes
tIPSecEncrypDecrypPkt consecutive_failure 6 platform_action 0 queue_priority las
t
#
#  GOLD TestIPSecEncrypDecrypPkt Test TCL script
#
#  March 2005, Hai Qiu
#
#  Copyright (c) 2005-2007 by cisco Systems, Inc.
#  All rights reserved.
#
#
# Register for TestIPSecEncrypDecrypPkt test even
# the elements for register the event
```

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

ap_perf_test_base_cpu.tcl Sample Policy

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

```
::cisco::eem::event_register_appl sub_system 798 type 9999
#----------------------------------
# EEM policy used for measuring the cpu performance of EEM policies.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005, 2006 by cisco Systems, Inc.
# All rights reserved.
#------------------
###
### Input arguments:
###
### arg1 $iter                    - current iteration count
###
### The following EEM environment variables are used:
###
### _perf_iterations (mandatory) - number of iterations over which we
###                                will run our measurement.
### Example:
### event manager environment _perf_iterations 100
###
### _perf_cmd1 (optional)        - optional non interactive cli command
###                                to be executed as part of the
###                                measurement test.
### Example:
### event manager environment _perf_cmd1 enable
###
### _perf_cmd2 (optional)        - optional non interactive cli command
###                to be executed as part of the
###                                measurement test.
###                                To use _perf_cmd2, _perf_cmd1 MUST
###                be defined.
### Example:
### event manager environment _perf_cmd2 show ver
###
### _perf_cmd3 (optional)        - optional non interactive cli command
###                                to be executed as part of the
###                                measurement test.
###                                To use _perf_cmd3, _perf_cmd1 MUST
###                                be defined.
### Example:
### event manager environment _perf_cmd3 show int counters protocol status
###
### Description:
###   Iterate through _perf_iterations of this policy.
###   It is up to the user to calculate the average
###   execution time based on the system timestamps.
###   Optional commands _perf_cmd1,
###   _perf_cmd2 and _perf_cmd3 are executed if defined.
###
###   A value of 100 is a good starting point.
###
### Outputs:
###   Console output.
###
### Usage example:
###   >conf t
###   >service timestamps debug datetime msec
###   >event manager environment _perf_iterations 100
###   >event manager policy ap_perf_base_cpu.tcl
###   >event manager policy no_perf_test_init.tcl
###   >end
###   2d19h: %SYS-5-CONFIG_I: Configured from console by console
###   >event manager run no_perf_test_init.tcl
###
###   Oct 16 14:57:17.284: %SYS-5-CONFIG_I: Configured from console by console
###   >event manager run no_perf_test_init.tcl
###
```

```
###   Oct 16 19:32:02.772: %HA_EM-6-LOG:
###     eem_policy/no_perf_test_init.tcl: EEM performance test start
###   Oct 16 19:32:03.115: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 1
###   Oct 16 19:32:03.467: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 2
###   ...
###   Oct 16 19:32:36.936: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test iteration 100
###   Oct 16 19:32:36.936: %HA_EM-6-LOG:
###     eem_policy/ap_perf_test_base_cpu.tcl: EEM performance test end
###
###   The user must calculate execution time and average time of execution.
###   In this example, total time = 19:32:36.936 - 19:32:02.772 = 34.164
###   Average script execution time = 341.64 milliseconds
###
# check if all the env variables we need exist
# If any of them doesn't exist, print out an error msg and quit
if {![info exists _perf_iterations]} {
    set result \
        "Policy cannot be run: variable _perf_iterations has not been set"
    error $result $errorInfo
}
# ensure our target iteration count > 0
if {$_perf_iterations <= 0} {
    set result \
        "Policy cannot be run: variable _perf_iterations <= 0"
    error $result $errorInfo
}
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# query the event info
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
set iter $arr_einfo(data1)
set iter [expr $iter + 1]
# if _perf_cmd1 is defined
if {[info exists _perf_cmd1]} {
    # open the cli library
    if [catch {cli_open} result] {
        error $result $errorInfo
    } else {
        array set cli1 $result
    }
    # execute the comamnd defined in _perf_cmd1
    if [catch {cli_exec $cli1(fd) $_perf_cmd1} result] {
        error $result $errorInfo
    }
    # if _perf_cmd2 is defined
    if {[info exists _perf_cmd2]} {
        # execute the comamnd defined in _perf_cmd2
        if [catch {cli_exec $cli1(fd) $_perf_cmd2} result] {
            error $result $errorInfo
        } else {
            set cmd_output $result
        }
    }
    # if _perf_cmd3 is defined
    if {[info exists _perf_cmd3]} {
        # execute the comamnd defined in _perf_cmd3
        if [catch {cli_exec $cli1(fd) $_perf_cmd3} result] {
            error $result $errorInfo
        } else {
            set cmd_output $result
        }
    }
    # close the cli library
    if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
        error $result $errorInfo
```

```
    }
}

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

tm_cli_cmd.tcl Sample Policy

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

```
::cisco::eem::event_register_timer cron name crontimer2 cron_entry $_cron_entry maxrun 240
#-------------------------------------------------------------------
# EEM policy that will periodically execute a cli command and email the
# results to a user.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------
### The following EEM environment variables are used:
###
### _cron_entry (mandatory)            - A CRON specification that determines
###                                      when the policy will run. See the
###                                      IOS Embedded Event Manager
###                                      documentation for more information
###                                      on how to specify a cron entry.
### Example: _cron_entry                 0-59/1 0-23/1 * * 0-7
###
### _log_file (mandatory without _email_....)
###                                    - A filename to append the output to.
###                                      If this variable is defined, the
###                                      output is appended to the specified
###                                      file with a timestamp added.
### Example: _log_file                   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
}
if [catch {cli_exec $cli1(fd) "en"} result] {
    error $result $errorInfo
}
```

```
# save exact execution time for command
set time_now [clock seconds]
# execute command
if [catch {cli_exec $cli1(fd) $_show_cmd} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
    # format output: remove trailing router prompt
    regexp {\n*(.*\n)([^\n]*)$} $result dummy cmd_output
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}

# 2. log the success of the CLI command
set msg [format "Command \"%s\" executed successfully" $_show_cmd]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# 3. if _log_file is defined, then attach it to the file
if {[info exists _log_file]} {
    # attach output to file
    if [catch {open $_log_file a+} result] {
        error $result
    }
    set fileD $result
    # save timestamp of command execution
    #      (Format = 00:53:44 PDT Mon May 02 2005)
    set time_now [clock format $time_now -format "%T %Z %a %b %d %Y"]
    puts $fileD "%%% Timestamp = $time_now"
    puts $fileD $cmd_output
    close $fileD
}
# 4. if _email_server is defined send the email out
if {[info exists _email_server]} {
    set routername [info hostname]
    if {[string match "" $routername]} {
 error "Host name is not configured"
    }
    if [catch {smtp_subst [file join $tcl_library email_template_cmd.tm]} \
     result] {
 error $result $errorInfo
    }
    if [catch {smtp_send_email $result} result] {
 error $result $errorInfo
    }
}
```

sl_intf_down.tcl Sample Policy

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

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

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

### The following EEM environment variables are used:
###
```

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

# check if all the env variables we need exist
# If any of them doesn't exist, print out an error msg and quit
if {![info exists _email_server]} {
    set result \
        "Policy cannot be run: variable _email_server has not been set"
    error $result $errorInfo
}
if {![info exists _email_from]} {
    set result \
        "Policy cannot be run: variable _email_from has not been set"
    error $result $errorInfo
}
if {![info exists _email_to]} {
    set result \
        "Policy cannot be run: variable _email_to has not been set"
    error $result $errorInfo
}
if {![info exists _email_cc]} {
     #_email_cc is an option, must set to empty string if not set.
     set _email_cc ""
}

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

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

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

set msg $arr_einfo(msg)
set config_cmds ""

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

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

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

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

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

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

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

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

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

Debugging Embedded Event Manager Policies Examples

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

Debugging the CLI Library

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

Example of the debug event manager tcl cli_library Command

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

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

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

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

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

Debugging the SMTP Library

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

Example of the debug event manager tcl smtp_library Command

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

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

```
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 220 XXXX.example.com ESMTP XXXX
 1.1.0; Tue,
25 Jun 2002 14:20:39 -0700 (PDT)
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : HELO XXXX.example.com
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 XXXX.example.com Hello
XXXX.example.com [XXXX],
pleased to meet you
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : MAIL FROM:<XX@example.com>
```

```
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Sender
 ok
00:39:46: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : RCPT TO:<XX@example.com>
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Recipient
 ok
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : RCPT TO:<XX@example.com>
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 250 <XX@example.com>... Recipient
 ok
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : DATA
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_read  : 354 Enter mail, end with "."
on a line by itself
00:39:47: tm_cli_cmd.tcl[0]: DEBUG(smtp_lib) : smtp_write : Date: 25 Jun 2002 14:35:00 UTC

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

Tracing Tcl set Command Operations Example

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

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

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

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

RPC Event Detector Example

```
TCL script (rpccli.tcl):
::cisco::eem::event_register_rpc
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
```

```
proc run_cli { clist } {
    set rbuf ""
    if {[llength $clist] < 1} {
    return -code ok $rbuf
    }
    if {[catch {cli_open} result]} {
        return -code error $result
    } else {
    array set cliarr $result
    }
    if {[catch {cli_exec $cliarr(fd) "enable"} result]} {
        return -code error $result
    }
    if {[catch {cli_exec $cliarr(fd) "term length 0"} result]} {
        return -code error $result
    }
    foreach cmd $clist {
    if {[catch {cli_exec $cliarr(fd) $cmd} result]} {
            return -code error $result
    }
    append rbuf $result
    }
    if {[catch {cli_close $cliarr(fd) $cliarr(tty_id)} result]} {
        puts "WARNING: $result"
    }
    return -code ok $rbuf
}
proc run_cli_interactive { clist } {
    set rbuf ""
    if {[llength $clist] < 1} {
    return -code ok $rbuf
    }
    if {[catch {cli_open} result]} {
        return -code error $result
    } else {
    array set cliarr $result
    }
    if {[catch {cli_exec $cliarr(fd) "enable"} result]} {
        return -code error $result
    }
    if {[catch {cli_exec $cliarr(fd) "term length 0"} result]} {
        return -code error $result
    }
    foreach cmd $clist {
        array set sendexp $cmd
    if {[catch {cli_write $cliarr(fd) $sendexp(send)} result]} {
            return -code error $result
    }
    foreach response $sendexp(responses) {
        array set resp $response
        if {[catch {cli_read_pattern $cliarr(fd) $resp(expect)} result]} {
                return -code error $result
        }
        if {[catch {cli_write $cliarr(fd) $resp(reply)} result]} {
                return -code error $result
        }
    }
    if {[catch {cli_read $cliarr(fd)} result]} {
            return -code error $result
    }
    append rbuf $result
    }
    if {[catch {cli_close $cliarr(fd) $cliarr(tty_id)} result]} {
        puts "WARNING: $result"
    }
    return -code ok $rbuf
}
array set arr_einfo [event_reqinfo]
set args $arr_einfo(argc)
set cmds [list]
for { set i 0 } { $i < $args } { incr i } {
    set arg "arg${i}"
    # Split each argument on the '^' character.  The first element is
```

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

Additional References

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

Related Documents

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

MIBs

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

RFCs

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

Technical Assistance

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

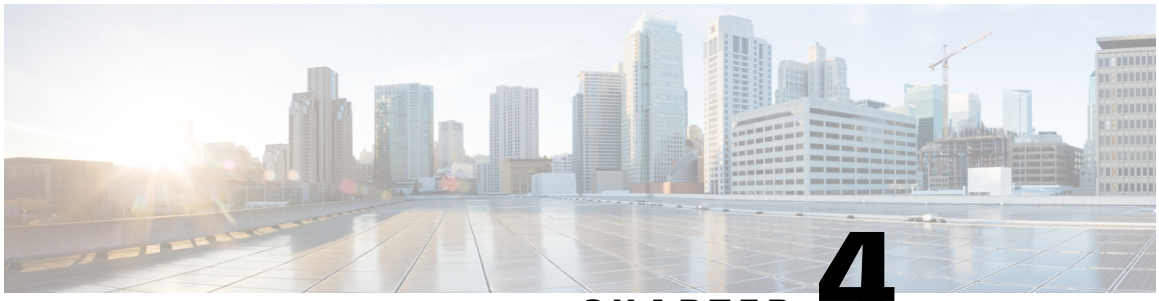
Feature Information for Writing Embedded Event Manager 3.2 Policies Using Tcl

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

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

Table 26: Feature Information for Writing Embedded Event Manager 3.2 Policies Using Tcl

Feature Name	Releases	Feature Information
Embedded Event Manager 3.2	Cisco IOS XE Release 3.3SE	EEM is a distributed and customized approach to event detection and recovery offered directly in a Cisco IOS device. In Cisco IOS XE Release 3.3SE, this feature is supported on Cisco Catalyst 3850 Series Switches and Cisco 5700 Wireless LAN Controllers. The following sections provide information about this feature: The following commands were introduced or modified: **debug event manager**, **event identity**, **event mat**, **event neighbor-discovery**, **show event manager detector**.



CHAPTER 4

EEM Action Tcl Command Extension

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:

priority low|normal|high

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.

- action_policy, page 172
- action_process, page 172
- action_program, page 174
- action_reload, page 174
- action_script, page 175
- action_snmp_trap, page 176
- action_snmp_object_value, page 176
- action_switch, page 177
- action_syslog, page 178
- action_track_read, page 178

- action_track_set, page 179

action_policy

Allows a Tcl script to run an Embedded Event Manager (EEM) policy that has been registered with the None event detector. The action of running an EEM policy can also be performed using the **event manager run** command.

Syntax

```
action_policy ?
```

Arguments

? (represents a string)	(Mandatory) The name of the EEM policy to be scheduled for execution. The policy must have been previously registered with the None event detector.

None

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 = 12)    FH_ENOSUCHEID (unknown event ID)
```
This error means that the policy is unknown because it is not registered.

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```
This error means that the action command requested was unknown.

action_process

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

Syntax

```
action_process start|restart|kill [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.
kill	(Mandatory) Specifies that a process is to be stopped (killed).
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) The pathname of a program to run.
argv	(Optional) The argument string of the program.
envp	(Optional) The environment string of the program.
stdin	(Optional) The pathname for stdin.
stdout	(Optional) The pathname for stdout.
stderr	(Optional) The 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_reload

Reloads the device.

Syntax

```
action_reload
```

Arguments

None

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.

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_snmp_trap

Sends a Simple Network Management Protocol (SNMP) trap using the Embedded Event Manager Notification MIB.

Syntax

```
action_snmp_trap [intdata1 ?] [intdata2 ?] [strdata ?]
```

Arguments

intdata1	(Optional) Arbitrary integer sent in trap.
intdata2	(Optional) Arbitrary integer sent in trap.
strdata	(Optional) Arbitrary string data sent in trap.

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.

action_snmp_object_value

Sets a Simple Network Management Protocol (SNMP) object ID and value to be returned for the SNMP get request.

Syntax

```
action_snmp_object_value {int|uint|counter|gauge|ipv4|octet|counter64|string} ?
[next_oid ?]
```

Arguments

int	A 32-bit number used to specify a numbered type within the context of a managed object.
uint	A 32-bit number used to represent decimal value.
counter	A 32-bit number with a minimum value of 0.
gauge	A 32-bit number with a minimum value of 0.
ipv4	IP version 4 address.
octet	An octet string in hex notation used to represent physical addresses.
counter 64	A 64-bit number with a minimum value of 0.
string	An octet string in text notation used to represent text strings.
next_oid	The OID of the next object in the table; NULL if it is the last object in the table.

Result String

None

Set _cerrno

Yes

action_switch

Switches processing to a secondary processor in a fully redundant environment. Before using the **action_switch** Tcl command extension, you must install a backup processor in the device. If the hardware is not fully redundant, the switchover action will not be performed.

Syntax

```
action_switch
```

Arguments

None

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.

action_syslog

Generates a periodic syslog message using the specified facility when an EEM script is triggered.

Syntax

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

Arguments

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

Result String

None

Set _cerrno

Yes

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 number)	(Mandatory) Tracked object number in the range from 1 to 500, inclusive.

Result String

```
number {%u}
state {%s}
```

Set _cerrno

Yes

FH_ENOTRACK
This error means that the tracked object number was not found.

action_track_set

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

Syntax

```
action_track_set ? state up|down
```

Arguments

? (represents a number)	(Mandatory) Tracked object number in the range from 1 to 500, inclusive.
state	(Mandatory) Specifies that the state of the tracked object will be set. If up is specified, the state of the tracked object will be set to up. If down is specified, the state of the tracked object will be set to down.

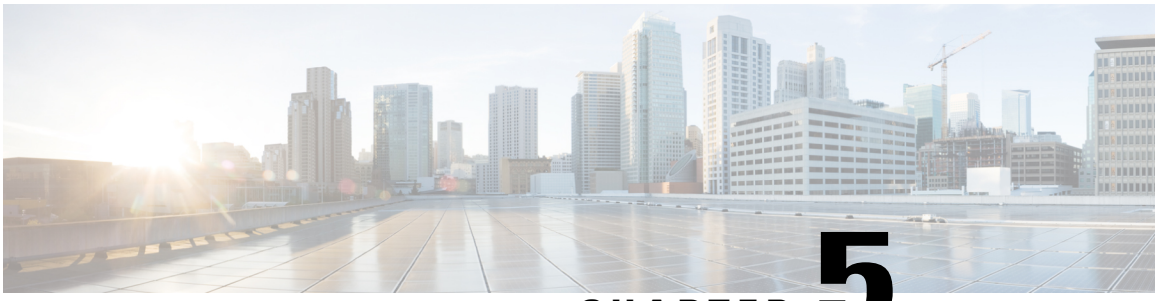
Result String

None

Set _cerrno

Yes

FH_ENOTRACK
This error means that the tracked object number was not found.



CHAPTER 5

EEM 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 device 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" section and in the "Using the CLI Library to Run an Interactive Command" section in the cli_write Tcl command.

The vty lines are allocated from the pool of vty lines that are configured using the **line vty** CLI configuration command. EEM will use a vty line when a vty line is not being used by EEM and there are available vty lines. EEM will also use a vty line when EEM is already using a vty line and there are three or more vty lines available. Be aware that the connection will fail when fewer than three vty lines are available, preserving the remaining vty lines for Telnet use.

Your release may support XML-PI. For details about the XML-PI support, the new CLI library command extensions, and some examples of how to implement XML-PI, see EEM CLI Library XML-PI Support.

- cli_close, page 182
- cli_exec, page 182
- cli_get_ttyname, page 183
- cli_open, page 183
- cli_read, page 184
- cli_read_drain, page 185
- cli_read_line, page 185
- cli_read_pattern, page 186

- cli_run, page 186
- cli_run_interactive, page 187
- cli_write, page 188

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

Allocates a vty, creates an EXEC command-line interface (CLI) session, and connects the vty to a channel handler. Returns an array including the channel handler.

Note Each call to **cli_open** initiates a Cisco IOS EXEC session that allocates a Cisco IOS vty line. The vty remains in use until the **cli_close** routine is called. The vty lines are allocated from the pool of vty lines that are configured using the **line vty** CLI configuration command. EEM will use a vty line when a vty line is not being used by EEM and there are available vty lines. EEM will also use a vty line when EEM is already using a vty line and there are three or more vty lines available. Be aware that the connection will fail when fewer than three vty lines are available, preserving the remaining vty lines 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 device prompt occurs in the contents read. Returns all the contents read up to the match.

Syntax

```
cli_read fd
```

Arguments

fd	(Mandatory) The CLI channel handler.

Result String

All the contents read.

Set _cerrno

Cannot get device name.

Note This Tcl command extension will block waiting for the device 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) The CLI channel handler.

Result String

The line read.

Set _cerrno

None

Note This Tcl command extension will block 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 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) The CLI channel handler.
ptn	(Mandatory) The 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 will block waiting for the specified pattern to show up in the contents read.

cli_run

Iterates over the items in the clist and assumes that each one is a command-line-interface (CLI) command to be executed in the enable mode. On success, returns the output of all executed commands and on failure, returns error from the failure.

Syntax

```
cli_run clist
```

Arguments

clist	(Mandatory) The list of commands to be executed.

Result String

Output of all the commands that are executed or an error message.

Set _cerrno

None.

Sample Usage

The following example shows how to use the **cli_run** command extension.

```
set clist [list {sh run} {sh ver} {sh event man pol reg}]
cli_run { clist }
```

cli_run_interactive

Provides a sublist to the clist which has three items. On success, returns the output of all executed commands and on failure, returns error from the failure. Also uses arrays when possible as a way of making things easier to read later by keeping expect and reply separated.

Syntax

```
cli_run_interactive clist
```

Arguments

clist	(Mandatory) List of three items:
	• command– Command to be executed
	• expect– A regular expression pattern match for the expected reply prompt
	• responses– A list of possible responses to the reply prompt constructed as an array of two items:
	  • expect– A regular expression pattern match for a possible reply prompt
	  • reply- A reply for that expected prompt

Result String

Output of all the commands that are executed or an error message. As each command is executed its output is appended to a result variable. Upon exhaustion of the input list, the CLI channel is closed and the aggregate result is returned.

Set _cerrno

None.

Sample Usage

The following example shows how to clear counters for interface fa0/0 use the cli_run_ interactive command extension.

```
set cmdarr(command) "clear counters fa0/0"
set cmdarr(responses) [list]
set resps(expect) {[confirm]}
set resps(reply) "y"
lappend cmdarr(responses) [array get resps]
set rc [catch {cli_run_interactive [list [array get cmdarr]]} result]
```

Possible errors raised include:

- cannot get pty for exec
- cannot spawn exec
- error reading the first prompt
- error reading the channel
- cannot close channel

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) "en"} result] {
puts stderr $result
exit 1
}
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 device 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 "en"} result] {
error $result $errorInfo
}
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 device prompt.

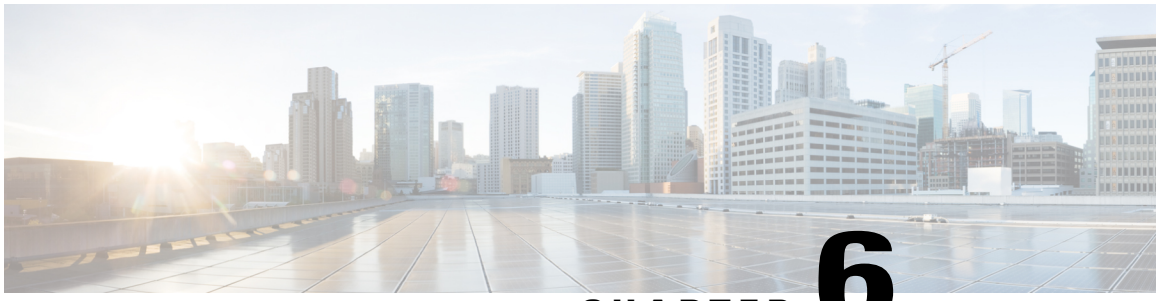
For example, use CLI commands to do squeeze bootflash: and save the output of this command in the Tcl variable cmd_output.

```
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "en"} result] {
error $result $errorInfo
}

# 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 device 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_exec $cli1(fd) "en"} result] {
    error $result $errorInfo
}
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
}
```



CHAPTER 6

EEM CLI Library XML-PI Support

XML Programmatic Interface (XML-PI) was introduced in Cisco IOS Release 12.4(22)T. XML-PI provides a programmable interface which encapsulates IOS command-line interface (CLI) show commands in XML format in a consistent way across different Cisco products. Customers using XML-PI will be able to parse IOS show command output from within Tcl scripts using well-known keywords instead of having to depend on the use of regular expression support to "screen-scrape" output.

The benefit of using the XML-PI command extensions is to facilitate the extraction of specific output information that is generated using a CLI **show** command. Most show commands return many fields within the output and currently a regular expression has to be used to extract specific information that may appear in the middle of a line. XML-PI support provides a set of Tcl library functions to facilitate the parsing of output from the IOS CLI format extension in the form of:

```
show
 <
show-command
> | format
 {
spec-file
}
```

where a spec-file is a concatenation of all Spec File Entries (SFE) for each **show** command currently supported. As part of the XML-PI project a default spec-file will be included in the IOS Release 12.4(22)T images. The default spec-file will have a small set of commands and the SFE for the commands will have a subset of the possible tags. If no spec-file is provided with the format command, the default spec-file is used.

For more general details about XML-PI, see the "XML-PI" module.

- xml_pi_exec, page 193
- xml_pi_parse, page 194
- xml_pi_read, page 195
- xml_pi_write, page 195

xml_pi_exec

Writes the XML-PI command specified using the cmd argument to the channel whose handler is specified using the fd argument and the spec-file specified by the spec_file argument to execute the command. The raw XML output data of the command is then read from the channel and the XML output is returned.

Syntax

```
xml_pi_show fd cmd [spec_file]
```

Arguments

fd	(Mandatory) The CLI library file descriptor obtained from cli_open.
cmd	(Mandatory) IOS show command.
spec_file	(Optional) IOS CLI show command spec_file.

Result String

Result of IOS show command in XML format.

Set _cerrno

Possible error raised:

1. error reading the channel

xml_pi_parse

Processes the XML show command raw output passed into this function as xml_data and retrieve those fields that are specified by xml_tags_list. The following processing occurs:

Step 1: The XML tag list is validated as a Tcl list. An XML tag can be specified as the low order XML tag name or as a fully qualified XML tag name in case the low order name is ambiguous for a given command.

Example tags: <Interface> <ShowIpInterfaceBrief><IPInterfaces><entry><Interface>

Step 2: The xml_data is validated as valid XML and parsed into an XML parse tree.

Step 3: A walk is made through the XML parse tree and each tag is compared with entries in the XML tag list. When a match occurs it is determined if the tag name matches a Tcl procedure defined within the current Tcl scope. If so, that Tcl procedure will be called with the current result. If not, the tag name and the data associated with that tag name will be appended to the current result.

Syntax

```
xml_pi_parse fd xml_show_cmd_output xml_tags_list
```

Arguments

fd	(Mandatory) The CLI library file descriptor obtained from cli_open.
xml_show_cmd_output	(Mandatory) Output of xml_pi_show command extension in xml format.

xml_tags_list	(Mandatory) List of interesting tags.

Result String

Data in a Tcl array indexed by XML tag name.

Note The current result is reset after Tcl procedure calls.

Set _cerrno

Possible errors raised:

1. error splitting the XML tags list 2. null XML tag list specified 3. XML tag tree exceeds 20 levels 4. called Tcl procedure returned an error 5. memory allocation failure 6. XML parse failure 7. failed to create XML domain

xml_pi_read

Reads the XML-PI command output (from the specified show command) from the CLI channel whose handler is given by the file descriptor until the pattern of the router prompt occurs in the contents that are read. Returns all the contents read up to the match in XML format.

Syntax

```
xml_pi_read fd
```

Arguments

fd	(Mandatory) The CLI library file descriptor obtained from cli_open.

Result String

All the contents that are read in XML format.

Set _cerrno

Possible errors raised:

1. cannot get router name 2. command error

xml_pi_write

Writes the XML-PI command specified using the cmd argument to the channel whose handler is given by the fd argument and the spec file specified by the spec_file argument.

Syntax

```
xml_pi_write fd cmd spec_file
```

Arguments

fd	(Mandatory) The CLI library file descriptor obtained from cli_open.
cmd	(Mandatory) IOS show command.
spec_file	(Optional) IOS CLI show command spec_file.

Result String

None

Set _cerrno

None

Sample Usage of the XML-PI feature

The following EEM policy (sample.tcl) presents one example that illustrates five different implementations of the new EEM XML-PI functionality. The odm spec-file (required for Example 2) follows this policy.

```
::cisco::eem::event_register_none maxrun 60
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# open the cli_lib.tcl channel
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}
# enter "enable" privilege mode
if [catch {cli_exec $cli1(fd) "en"} result] {
error $result $errorInfo
}
# Example 1:
#
# Detect if XML-PI is present in this image
# Invoke xml_pi_exec with the default spec file for the "show inventory"
# command. After the command executes $result contains the raw XML data if
# the command is successful.
if [catch {xml_pi_exec $cli1(fd) "show inventory" ""} result] {
puts "Example 1: XML-PI support is not present in this image - exiting"
exit
} else {
puts "Example 1: XML-PI support is present in this image"
}
# Example 2:
#
# In the next example we demonstrate how to extract two data elements
# from the "show version" command using the specified XML-PI spec file.
# The raw output from this command is as follows:
#
# router#show version | format disk2:speceemtest.odm
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowVersion>
# <Version>12.4(20071029:194217)</Version>
```

```
# <Compiled>Thu 08-Nov-07 11:28</Compiled>
# <ROM>System Bootstrap, Version 12.2(20030826:190624) [BLD-npeg1_rommon_r11 102],
DEVELOPMENT</ROM>
# <uptime>17 minutes</uptime>
# <processor>NPE-G1</processor>
# <bytesofmemory>983040K/65536K</bytesofmemory>
# <CPU>700MHz</CPU>
# <L2Cache>0.2</L2Cache>
# <GigabitEthernetinterfaces>3</GigabitEthernetinterfaces>
# <bytesofNVRAM>509K</bytesofNVRAM>
# <bytesofATAPCMCIAcard>125952K</bytesofATAPCMCIAcard>
# <Sectorsize>512 bytes</Sectorsize>
# <bytesofFlashinternalSIMM>16384K</bytesofFlashinternalSIMM>
# <Configurationregister>0x2100</Configurationregister>
# </ShowVersion>
#
# Invoke xml_pi_exec with the spec file "disk2:speceemtest.odm" for the
# "show version" command. After the command executes $result contains
# the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show version" "disk2:speceemtest.odm"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <processor> and <CPU> fields be returned.
array set xml_result [xml_pi_parse $cli1(fd) $result "<processor> <CPU>"]
puts "Example 2: Processor is $xml_result(<processor>) CPU is $xml_result(<CPU>)"
}
# Example 3:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show inventory" command using the default built-in
# XML-PI spec file. Sample raw output from this command is as follows:
#
# router#show inventory | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowInventory>
# <SpecVersion>built-in</SpecVersion>
# <InventoryEntry>
# <ChassisName>&quot;Chassis&quot;</ChassisName>
# <Description>&quot;Cisco 7206VXR, 6-slot chassis&quot;</Description>
# <PID>CISCO7206VXR</PID>
# <VID>
# </VID>
# <SN>31413378 </SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>&quot;NPE-G1 0&quot;</ChassisName>
# <Description>&quot;Cisco 7200 Series Network Processing Engine
NPE-G1&quot;</Description>
# <PID>NPE-G1</PID>
# <VID>
# </VID>
# <SN>31493825 </SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>&quot;disk2&quot;</ChassisName>
# <Description>&quot;128MB Compact Flash Disk for NPE-G1&quot;</Description>
# <PID>MEM-NPE-G1-FLD128</PID>
# <VID>
# </VID>
# <SN>NAME: &quot;module 1&quot;</SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>&quot;module 1&quot;</ChassisName>
# <Description>&quot;Dual Port FastEthernet (RJ45)&quot;</Description>
# <PID>PA-2FE-TX</PID>
# <VID>
# </VID>
# <SN>JAE0827NGKX</SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>&quot;Power Supply 2&quot;</ChassisName>
```

```
# <Description>&amp;quot;Cisco 7200 AC Power Supply&amp;quot;</Description>
# <PID>PWR-7200-AC</PID>
# <VID>
# </VID>
# </InventoryEntry>
# </ShowInventory>
#
# Define a procedure to be called every time the <InventoryEntry> tag
# is processed. Since this tag precedes each new output record, the data
# that is passed into this procedure contains the fields that have been
# requested via xml_pi_parse since the previous time this procedure was
# called.
proc <InventoryEntry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
set output [format "Example 3: Item %2d %-18s %s" \
$num $xml_result(<PID>) $xml_result(<Description>)]
puts $output
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show inventory" command. After the command executes $result contains
# the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show inventory"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <PID> and <Description> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <InventoryEntry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<InventoryEntry> <PID> <Description>"]
# Display the data from the last record.
incr num
set output [format "Example 3: Item %2d %-18s %s" \
$num $xml_result(<PID>) $xml_result(<Description>)]
puts $output
}
# Example 4:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show ip interface brief" command using the default
# built-in XML-PI spec file. Sample raw output from this command is as
# follows:
#
# router#show ip interface brief | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowIpInterfaceBrief>
# <SpecVersion>built-in</SpecVersion>
# <IPInterfaces>
# <entry>
# <Interface>GigabitEthernet0/1</Interface>
# <IP-Address>172.19.209.34</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>up</Status>
# <Protocol>up</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/2</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
```

```
# </entry>
# <entry>
# <Interface>GigabitEthernet0/3</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/0</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/1</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# </IPInterfaces>
# </ShowIpInterfaceBrief>
#
# Define a procedure to be called every time the fully qualified name
# <ShowIpInterfaceBrief><IPInterfaces><entry> tag is processed. Since
# this tag precedes each new output record, the data that is passed into
# this procedure contains the fields that have been requested via
# xml_pi_parse since the previous time this procedure was called.
proc <ShowIpInterfaceBrief><IPInterfaces><entry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
set output [format "Example 4: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
} else {
puts "Example 4: Display All Interfaces"
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show ip interface brief" command. After the command executes $result
# contains the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show ip interface brief"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <Interface> and <Status> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <entry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<ShowIpInterfaceBrief><IPInterfaces><entry> <Interface> <Status>"]
# Display the data from the last record.
incr num
set output [format "Example 4: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
}
# Example 5:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show ip interface brief" command using the default
```

```
# built-in XML-PI spec file. Sample raw output from this command is as
# follows:
#
# router#show ip interface brief | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowIpInterfaceBrief>
# <SpecVersion>built-in</SpecVersion>
# <IPInterfaces>
# <entry>
# <Interface>GigabitEthernet0/1</Interface>
# <IP-Address>172.19.209.34</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>up</Status>
# <Protocol>up</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/2</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/3</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/0</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/1</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# </IPInterfaces>
# </ShowIpInterfaceBrief>
#
# Note: This example is the same as Example 4 with the exception that
# the new record procedure is called by the un-qualified tag name. The
# ability to specify the un-qualified tag names is simplier but only works
# if the un-qualified name is used once per Tcl program. In this example
# the unqualified new record tag name is "<entry>" which is a very
# common name in the Cisco spec file.
# Define a procedure to be called every time the <entry> tag
# is processed. Since this tag precedes each new output record, the data
# that is passed into this procedure contains the fields that have been
# requested via xml_pi_parse since the previous time this procedure was
# called.
proc <entry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
if ([string equal $xml_result(<Status>) "up"]) {
set output [format "Example 5: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
```

```
}
} else {
puts "Example 5: Display All Interfaces That Are Up"
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show ip interface brief" command. After the command executes $result
# contains the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show ip interface brief"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <Interface> and <Status> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <entry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<entry> <Interface> <Status>"]
# Display the data from the last record.
incr num
if ([string equal $xml_result(<Status>) "up"]) {
set output [format "Example 5: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
}
}
```

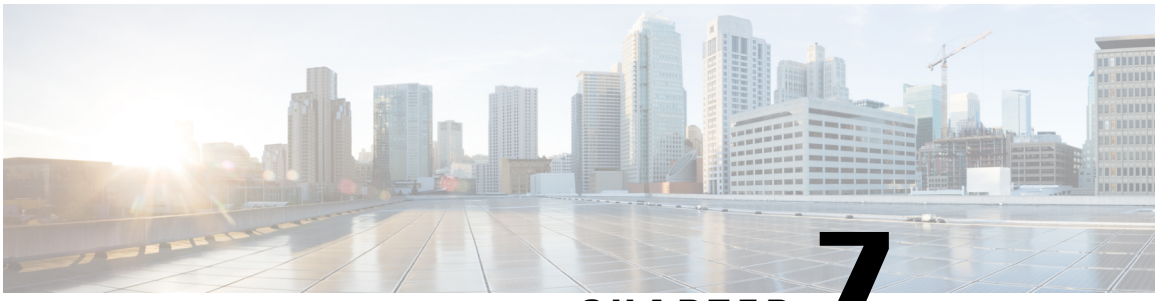
Sample XML-PI spec eemtest.odm ODM File:

```
###
show version
<?xml version='1.0' encoding='utf-8'?>
<ODMSpec>
<Command>
<Name>show version</Name>
</Command>
<OS>ios</OS>
<DataModel>
<Container name="ShowVersion">
<Property name="Version" distance = "1.0" length = "1" type = "IpAddress"/>
<Property name="Technical Support" distance = "1.0" length = "1" type = "IpAddress"/>
<Property name="Compiled" distance = "1.0" length = "3" type = "String"/>
<Property name="ROM" distance = "1.0" length = "7" type = "IpAddress"/>
<Property name="uptime" distance = "2" length = "8" type = "String"/>
<Property name="image" distance = "4" length = "1" type = "IpAddress"/>
<Property name="processor" distance = "-1" length = "1" type = "String"/>
<Property name="bytes of memory" distance = "-1" length = "1" type = "Port"/>
<Property name="CPU" distance = "2" length = "1" end-delimiter = "," type = "String"/>
<Property name="L2 Cache" distance = "-2" length = "1" end-delimiter = "," type = "String"/>
<Property name="Gigabit Ethernet interfaces" distance = "-1" length = "1" type = "Integer"/>
<Property name="bytes of NVRAM" distance = "-1" length = "1" type = "String"/>
<Property name="bytes of ATA PCMCIA card" distance = "-1" length = "1" type = "String"/>
<Property name="Sector size" distance = "1.0" length = "2" end-delimiter = ")" type =
"String"/>
<Property name="bytes of Flash internal SIMM" distance = "-1" length = "1" type = "String"/>
<Property name="Configuration register" distance = "2" length = "1" type = "String"/>
</Container>
</DataModel>
</ODMSpec>
```

Example sample.tcl Run:

```
router#config t
Enter configuration commands, one per line. End with CNTL/Z.
router(config)#event manager policy sample.tcl
router(config)#end
```

```
router#
Oct 10 20:21:26: %SYS-5-CONFIG_I: Configured from console by console
router#event manager run sample.tcl
Example 1: XML-PI support is present in this image
Example 2: Processor is NPE-G1 CPU is 700MHz
Example 3: Item 1 CISCO7206VXR "Cisco 7206VXR, 6-slot chassis"
Example 3: Item 2 NPE-G1 "Cisco 7200 Series Network Processing Engine NPE-G1"
Example 3: Item 3 MEM-NPE-G1-FLD128 "128MB Compact Flash Disk for NPE-G1"
Example 3: Item 4 PA-2FE-TX "Dual Port FastEthernet (RJ45)"
Example 3: Item 5 PWR-7200-AC "Cisco 7200 AC Power Supply"
Example 4: Display All Interfaces
Example 4: Interface 1 GigabitEthernet0/1 up
Example 4: Interface 2 GigabitEthernet0/2 administratively down
Example 4: Interface 3 GigabitEthernet0/3 administratively down
Example 4: Interface 4 FastEthernet1/0 administratively down
Example 4: Interface 5 FastEthernet1/1 administratively down
Example 4: Interface 6 SSLVPN-VIF0 up
Example 5: Display All Interfaces That Are Up
Example 5: Interface 1 GigabitEthernet0/1 up
Example 5: Interface 6 SSLVPN-VIF0 up
```



CHAPTER 7

EEM Context Library Command Extensions

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

- context_retrieve, page 203
- context_save, page 206

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) The array index.

Note The index_if_array argument will be 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) The 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

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"
}
```

CHAPTER 8

EEM Event Registration Tcl Command Extensions

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:

priority low|normal|high

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.

- event_register_appl, page 212
- event_register_cli, page 215
- event_register_counter, page 219
- event_register_gold, page 221
- event_register_identity, page 229
- event_register_interface, page 231
- event_register_ioswdsysmon, page 238
- event_register_ipsla, page 242
- event_register_mat, page 246
- event_register_neighbor_discovery, page 247

- event_register_nf, page 252
- event_register_none, page 256
- event_register_oir, page 258
- event_register_process, page 260
- event_register_resource, page 264
- event_register_rf, page 266
- event_register_routing, page 269
- event_register_rpc, page 272
- event_register_snmp, page 275
- event_register_snmp_notification, page 280
- event_register_snmp_object, page 283
- event_register_syslog, page 286
- event_register_timer, page 290
- event_register_timer_subscriber, page 295
- event_register_track, page 298
- event_register_wdsysmon, page 300

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.

In order to register for an application event, a subsystem must be specified. Either a Tcl policy or the internal Embedded Event Manager (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 [tag ?] sub_system ? type ? [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.

sub_system	(Mandatory) 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	(Mandatory) 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 in order for the publishing and registration to work.
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 exist, the application event will be raised when all the conditions are satisfied.

Result String

None

Set _cerrno

No

Event_reqinfo

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the Embedded Event Manager (EEM).
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.
data1 data2 data3 data4	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.

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.

Note The user 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.

Note The functionality provided in the CLI event detector only allows a regular expression pattern match on a valid IOS CLI command itself. This does not include text after a pipe character when redirection is used.

Syntax

```
event_register_cli [tag ?] sync yes|no skip yes|no
[occurs ?] [period ?] pattern ? [default ?] [enter] [questionmark] [tab] [mode]
[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.
sync	(Mandatory) A "yes" means that the policy (the event publish) will run synchronously with the CLI command; a "no" means that the event publish will be performed asynchronously with the CLI command. The event detector will be notified when the policy completes running. The exit status of the policy indicates whether or not the CLI command should be executed: if the exit status is zero, which means that the policy is executed successfully, the CLI command will not be executed; otherwise, the CLI command will be executed.
skip	Mandatory if the sync argument is "no" and should not exist if the sync argument is "yes." If the skip argument is "yes," it means that the CLI command should not be executed. If the skip argument is "no," it means that the CLI command should be executed. **Caution** When the skip argument is "yes," unintended results may be produced if the pattern match is made for configuration commands because the CLI command that matches the regular expression will not be executed.

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.

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.
enter	(Optional) Specifies to perform the event match when the user presses the Enter key. When this parameter is used, the input string will not be expanded before matching.
questionmark	(Optional) Specifies to perform the event match when the user presses the ? key. When this parameter is used, the input string will not be expanded before matching.
tab	(Optional) Specifies to perform the event match when the user presses the Tab key. When this parameter is used, the input string will not be expanded before matching.
mode	(Optional) Events will only be generated when the parser is in the specified parser mode. The available modes can be listed using the **show parser dump** CLI command. The mode parameter is checked when any one of the optional parameters--enter, questionmark, or tab-- is specified.

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 CLI event will be raised when all the conditions are matched.

Result String

None

Set _cerrno

No

Note This policy runs before the CLI command is executed. For example, suppose policy_CLI is registered to run when the **copy** command is entered. When the **copy** command is entered, the CLI event detector finds a pattern match and triggers this policy to run. When the policy execution ends, the CLI event detector determines if the **copy** command needs to be executed according to "sync", "skip" (set in the policy), and the exit status of the policy execution if needed.

Event_reqinfo

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u msg {%s} msg_count %d line %u key %u tty %u error_code %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, at which the event was published to the EEM.
event_severity	The severity of the event.

Event Type	Description
msg	Text entered at the CLI prompt.
msg_count	Number of times the pattern matched before the event was triggered.
line	The text the parser was able to expand up to the point where the matched key was entered.
key	The enter, questionmark, or tab key.
tty	Corresponds to the line number the user is executing the command on.
error_code	The error code in CLI. 0 --No error from parser up to point where a key was entered. 1--Command is ambiguous up to point where a key was entered. 4--Unknown command up to point where a key was entered.

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 [tag ?] 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|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.
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 will be raised and event monitoring will be 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 will be 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: • 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.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
name	Counter name.

event_register_gold

Registers for a Generic Online Diagnostic (GOLD) failure event. Use this Tcl command extension to run a policy on the basis of a Generic Online Diagnostic (GOLD) failure event for the specified card and subcard.

Syntax

```
event_register_gold card all|card_number
[subcard all|subcard_number]
[new_failure TRUE|FALSE]
[severity_major TRUE]
[severity_minor TRUE]
[severity_normal TRUE]
[action_notify TRUE|FALSE]
[testing_type [bootup|ondemand|schedule|monitoring]]
[test_name [testname]]
[test_id [testnumber]]
```

```
[consecutive_failure consecutive_failure_number]
[platform_action [action_flag]]
[maxrun ?]
[queue_priority low|normal|high|last]
[nice 0|1]
```

Arguments

card	(Mandatory) Specifies whether all cards or one card is to be monitored: • card all--Specifies that all cards are to be monitored. This is the default. • card-number--Specifies that the card identified by the number card-number is to be monitored. This argument must be specified to complete the **event_register_gold**Tcl command extension.
subcard	(Optional) Specifies that one or more subcards are to be monitored: • subcard all--Specifies that all subcards are to be monitored. • subcard-number--Specifies that the subcard identified by the number subcard-number is to be monitored. If this argument is not specified, all subcards are monitored by default.
new_failure	(Optional) Specifies event criteria based on the new test failure information from GOLD: • new_failure TRUE--Specifies that the event criterion for the new test failure is true from GOLD. • new_failure FALSE--Specifies that the event criterion for the new test failure is false from GOLD. If this argument is not specified, the new test failure information from GOLD is not considered in the event criteria.
severity_major	(Optional) Specifies that the event criteria for diagnostic result matches with the diagnostic major error from GOLD.
severity_minor	(Optional) Specifies that the event criteria for diagnostic result matches with diagnostic minor error from GOLD.

severity_normal	(Optional) Specifies that the event criteria for diagnostic result matches with diagnostic normal from GOLD. This is the default.
action_notify	(Optional) Specifies the event criteria based on the action notify information from GOLD: • action_notify TRUE--Specifies that the event criterion for the action notify is true from GOLD. • action_notify FALSE--Specifies that the event criterion for the action notify is false from GOLD. If this argument is not specified, the action notify information from GOLD is not considered in the event criteria.
testing_type	(Optional) Specifies the event criteria based on the testing types of the diagnostic from GOLD: • testing_type bootup--Specifies the diagnostic tests that are running on system bootup. • testing_type ondemand--Specifies the diagnostic tests that are running from CLI after the card is online. • testing_type schedule--Specifies the scheduled diagnostic tests. • testing_type monitoring--Specifies the diagnostic tests that are running periodically in the background to monitor the health of the system. If this argument is not specified, the testing type information from GOLD is not considered in the event criteria and the policy applies to all the diagnostic testing types.
test_name	(Optional) Specifies the event criteria based on the test name: • test_name test-name--Specifies the event criteria based on the test with the name test-name. If this argument is not specified, the test name information from GOLD is not considered in the event criteria.

test_id	(Optional) Specifies the event criteria based on test ID: • test_id test-id--Specifies the event criteria based on the test with the ID number test-id. The maximum value of test-id is 65535. **Note** Because the test ID can be different for the same test on different line cards, usually the test_name keyword should be used instead. If the test ID is specified and conflicts with the specified test name, the test name overwrites the test ID. If this argument is not specified, test ID information from GOLD is not considered in the event criteria.
consecutive_failure	(Optional) Specifies the event criteria based on consecutive test failure information from GOLD: • consecutive_failure consecutive-failure-number--Specifies that the event criterion is based on the occurrence of consecutive-failure-number consecutive test failures. If this argument is not specified, consecutive test failure information from GOLD is not considered in the event criteria.
platform_action	(Optional) Specifies whether callback to the platform is needed when all the event criteria are matched. When callback is needed, the platform needs to register a callback function through the provided registry. • platform_action action-flag-number--Specifies that, when callback to the platform is needed, specific information is specified by the platform-specific action-flag-number value. The maximum value of action-flag-number is 65535. **Note** It is up to the platform to determine what action needs to be taken based on the flag. If this argument is not specified, there is no callback.

maxrun	(Optional) Specifies the maximum runt time of the script. • maxrun max-run-time-number--Specifies that the maximum run time of the script is max-run-time-number seconds. The maximum value of max-run-time-number is 4294967295 seconds. If this argument is not specified, the default run time is 20 seconds.
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.
nice	(Optional) Policy run-time priority setting: • nice 0--Specifies that the policy is run at the default run-time priority level. • nice 1--Specifies that the policy is run at a run-time priority that is less than the default priority. If this argument is not specified, the default run-time priority is used.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"event_id %u event_type %u event_type_string {%s} %u card %u sub_card %u"
"event_severity {%s} event_pub_sec %u event_pub_msec %u overall_result %u"
"new_failure {%s} action_notify {%s} tt %u tc %u bl %u ci %u pc %u cn {%s}"
"sn {%s} tn# {%s} ta# %s ec# {%s} rc# %u lf# {%s} tf# %u cf# %u tr# {%s}"
"tr#p# {%s} tr#d# {%s}"
```

Event Type	Description
action_notify	Action notify information in GOLD event: true or false.
bl	The boot-up diagnostic level, which can be one of the following values: • 0: complete diagnostic • 1: minimal diagnostics • 2: bypass diagnostic
card	Card information for the GOLD event.
cf *testnum*	Consecutive failure, where *testnum* is the test number. For example, **cf3** is the EEM built-in environment variable for consecutive failure of test 3.
ci	Card index.
cn	Card name.
ec *testnum*	Test error code, where *testnum* is the test number. For example, **ec3** is the EEM built-in environment variable for the error code of test 3.
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_pub_msec event_pub_sec	The time, in milliseconds and seconds, when the event was published to the EEM.

Event Type	Description
event_severity	GOLD event severity, which can be one of the following values: • normal • minor • major.
event_type	Type of event.
event_type_string	An ASCII string that represents the name of the event for this event type.
lf *testnum*	Last fail time, where *testnum* is the test number. For example, **lf3** is the EEM built-in variable for the last fail time of test 3. The timestamp format is *mmm dd yyyy hh*:*mm*:*ss*. For example, Mar 11 1960 08:47:00.
new_failure	The new test failure information in a GOLD event flag: true or false.
overall_result	The overall diagnostic result, which can be one of the following values: • 0: OK • 3: minor error • 4: major error • 14: unknown result
pc	Port counts.
rc *testnum*	Test total run count, where *testnum* is the test number. For example, **rc3** is the EEM built-in variable for the total run count of test 3.
sn	Card serial number.
sub_card	The subcard on which a GOLD failure event was detected.
ta *testnum*	Test attribute, where *testnum* is the test number. For example, **ta3** is the EEM built-in variable for the test attribute of test 3.
tc	Test counts.

Event Type	Description
tf *testnum*	Total failure count, where *testnum* is the test number. For example, **tf3** is the EEM built-in variable for the total failure count of test 3.
tn *testnum*	Test name, where *testnum* is the test number. For example, **tn3** is the EEM built-in variable for the name of test 3.
tr *testnum*	Test result, where *testnum* is the test number. For example, **tr6** is the EEM built-in variable for test 6 where test 6 is not a per-port test and not a per-device test. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown
tr *testnum* **d** *devnum*	Per-device test result, where *testnum* is the test number and *devnum* is the device number. For example, **tr3d20** is the EEM built-in variable for the test result for test 3, device 20. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown
tr *testnum* **p** *portnum*	Per-port test result, where *testnum* is the test number and *portnum* is the device number. For example, **tr5p20** is the EEM built-in variable for the test result for test 3, port 20. The test result is one of the following values: • P: diagnostic result Pass • F: diagnostic result Fail • U: diagnostic result Unknown

Event Type	Description
tt	The testing type, which can be one of the following: • 1: A boot-up diagnostic • 2: An on-demand diagnostic • 3: A schedule diagnostic • 4: A monitoring diagnostic

event_register_identity

Registers for an identity event. Use this Tcl command extension to generate an event when AAA authentication or authorization is successful or failure or after normal user traffic on the port is allowed to flow.

Syntax

```
event_register_identity [tag ?] interface ?
[aaa-attribute ?]
[authc {all | fail | success}]
[authz {all | fail | success}]
[authz-complete]
[mac-address ?]
[queue_priority {normal | low | 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.
interface	A regular expression pattern to match against interface names.
aaa-attribute	(Optional) A regular expression that can be used to filter events by specific AAA attributes.
authc	(Optional) Triggers events on successful, failed or both successful and failed authentication.
authz	(Optional) Triggers events on successful, failed or both successful and failed authorization.
authz-complete	(Optional) Triggers events once the device connected to the interface is fully authenticated, authorized and normal traffic has begun to flow on that interface.

mac-address	(Optional) A regular expression pattern that can be used to filter events by mac addresses of the remote device.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 31536000, 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.
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. 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.
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_reqinfo For EEM_EVENT_IDENTITY

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u identity_stage %u identity_status %u interface %u identity_mac %u
identity_<attribute> {%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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, at which the event was published to the EEM.
event_severity	The severity of the event.
identity_stage	One among authentication, autherization or autherization-complete stages.
identity_status	Success or one of these failure types: fail_authc, fail_aaa_server, fail_no_response, fail_timeout, fail_authz. For autherization-complete it is always success.
interface	The interface for the event.
identity_mac	The MAC address of the remote device for the event.
identity_<attribute>	For each AAA attribute, a set a dynamic variable to the value corresponding to that AAA attribute in the attribute or value list.

event_register_interface

Registers for an interface counter event. Use this Tcl command extension to generate an event when specified interface counters exceed specified thresholds.

Syntax

```
event_register_interface [tag ?] name ?
parameter ? entry_op gt|ge|eq|ne|lt|le
entry_val ? entry_val_is_increment TRUE|FALSE
entry_type value|increment|rate
[exit_comb or|and]
[exit_op gt|ge|eq|ne|lt|le]
```

```
[exit_val ?] [exit_val_is_increment TRUE|FALSE]
[exit_type value|increment|rate]
[exit_time ?] [poll_interval ?]
[average_factor ?] [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.
name	(Mandatory) The name of the interface being monitored, for example, Ethernet 0/0. Abbreviations and spaces are not allowed.
parameter	(Mandatory) The name of the counter being compared as follows: • input_errors--Includes runts, giants, no buffer, CRC, frame, overrun, and ignored counts. Other input-related errors can also cause the input errors count to be increased, and some datagrams may have more than one error; therefore, this sum may not balance with the sum of enumerated input error counts. • input_errors_crc--Cyclic redundancy checksum generated by the originating LAN station or far-end device does not match the checksum calculated from the data received. • input_errors_frame--Number of packets received incorrectly having a CRC error and a noninteger number of octets. • input_errors_overrun--Number of times the receiver hardware was unable to hand received data to a hardware buffer because the input rate exceeded the receiver's ability to handle the data. • input_packets_dropped--Number of packets dropped because of a full input queue. • interface_resets--Number of times that an interface has been completely reset. • output_buffer_failures--Number of failed buffers and number of buffers swapped out. • output_buffer_swappedout--Number of packets swapped to DRAM.

parameter (continued)	• output_errors--Sum of all errors that prevented the final transmission of datagrams out of the interface being examined. Note that this may not balance with the sum of the enumerated output errors, because some datagrams may have more than one error, and others may have errors that do not fall into any of the specifically tabulated categories. • output_errors_underrun--Number of times that the transmitter has been running faster than the device can handle. • output_packets_dropped--Number of packets dropped because of a full output queue. • receive_broadcasts--Number of broadcast or multicast packets received by the interface. • receive_giants--Number of packets that are discarded because they exceed the maximum packet size of the medium. • receive_rate_bps--Interface receive rate in bytes per second. • receive_rate_pps--Interface receive rate in packets per second. • receive_runts--Number of packets that are discarded because they are smaller than the minimum packet size of the medium. • receive_throttle--Number of times that the receiver on the port was disabled, possibly because of buffer or processor overload. • reliability--Reliability of the interface as a fraction of 255 (255/255 is 100 percent reliability), calculated as an exponential average over 5 minutes. • rxload--Receive rate of the interface as a fraction of 255 (255/255 is 100 percent). • transmit_rate_bps--Interface transmit rate in bytes per second. • transmit_rate_pps--Interface transmit rate in packets per second. • txload--Transmit rate of the interface as a fraction of 255 (255/255 is 100 percent).

entry_op	(Mandatory) The comparison operator used to compare the current interface value with the entry value; if true, an event will be raised and event monitoring will be disabled until exit criteria are met.
entry_val	(Mandatory) The value at which the event will be triggered.
entry_val_is_increment	(Mandatory) If TRUE, the entry_val field is treated as an incremental difference and is compared with the difference between the current counter value and the value when the event was last true (the first polled sample if this is a new event). A negative value checks the incremental difference for a counter that is decreasing. If FALSE, the entry_val field is compared against the current counter value. **Note** This keyword has been deprecated, and if specified, the syntax is converted into equivalent entry-type keyword syntax.
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-valis 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) Used to indicate the combination of exit condition tests required to rearm the event trigger; if the and operator is specified, both exit value and exit time tests must be true to cause rearm; if the or operator is specified, either exit value or exit time tests can be true to cause event monitoring to be rearmed.

exit_op	(Optional) The comparison operator used to compare the current interface value with the exit value; if true, event monitoring for this event will be reenabled.
exit_val	(Optional) The value at which the event is rearmed to be monitored again.
exit_val_is_increment	(Optional) If TRUE, the exit_val field is treated as an incremental difference and is compared with the difference between the current counter value and the value when the event was last true. A negative value checks the incremental difference for a counter that is decreasing. If FALSE, the exit_val field is compared against the current counter value. **Note** In Cisco IOS Release 12.4(20)T, this keyword is deprecated, and if specified, the syntax is converted into equivalent exit-type keyword syntax.
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) The time period at which the event is rearmed to be monitored again (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).

poll_interval	(Optional) The frequency used to collect the samples (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 60 and 4294967295, inclusive, and where MMM must be an integer representing milliseconds between 0 and 999). The poll interval value must not be less than 1 second. The default is 1 second.
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.
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.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"event_severity {%s} name {%s} parameter {%s} value %d"
```

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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	Interface event severity, which can be one of the following values: • normal • minor • major
name	Name of the interface.
parameter	Name of the parameter.
value	The incremental/decremental difference compared to the last event triggered or the absolute value of the parameter being monitored, depending on the specified value of entry_val_is_increment.

event_register_ioswdsysmon

Registers for an IOSWDSysMon event. Use this Tcl command extension to generate an event when a Cisco IOS task exceeds specific CPU utilization or memory thresholds. A Cisco IOS task is called a Cisco IOS process in native Cisco IOS.

Syntax

```
event_register_ioswdsysmon [tag ?] [timewin ?] [sub12op and|or] [sub1 ?] [sub2 ?]
[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.
timewin	(Optional) Defines the time window within which all of the subevents must occur in order for an event to be generated (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).
sub12_op	(Optional) The combination operator for comparison between subevent 1 and subevent 2.
sub1	(Optional) The subevent 1 specification.
sub2	(Optional) The subevent 2 specification.

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 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). 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.

Subevent Syntax

```
cpu_proc path ? taskname ? op gt|ge|eq|ne|lt|le val ? [period ?]
mem_proc path ? taskname ? op gt|ge|eq|ne|lt|le val ? [is_percent TRUE|FALSE] [period ?]
```

Subevent Arguments

cpu_proc	(Mandatory) Specifies the use of a sample collection of CPU statistics.

path	(Mandatory) Software Modularity images only. The pathname of the POSIX process that contains the Cisco IOS scheduler to be monitored. For example, /sbin/cdp2.iosproc.
taskname	(Mandatory) The name of the Cisco IOS task to be monitored.
op	(Mandatory) The comparison operator used to compare the collected usage sample with the specified value; if true, an event will be raised.
val	(Mandatory) The value to be compared.
period	(Optional) The elapsed time period for the collection samples to be averaged (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 sample is used.
mem_proc	(Mandatory) Specifies the use of a sample collection of memory statistics.
is_percent	(Optional) Whether the specified value is a percentage.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"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	Description
event_type_string	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
num_subs	Number of subevents.

Where the subevent info string is for a CPU_UTIL subevent,

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

Subevent Type	Description
type	Type of subevent.
procname	POSIX process name for this subevent.
pid	POSIX process ID for this subevent.
taskname	Cisco IOS task name for this subevent.
taskid	Cisco IOS task ID for this subevent.
value	Actual average CPU utilization over the measured interval.
sec , **msec**	Elapsed time period for this measured interval.

Where the subevent info string is for a MEM_UTIL subevent,

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

Subevent Type	Description
type	Type of subevent.
procname	POSIX process name for this subevent.
pid	POSIX process ID for this subevent.
taskname	Cisco IOS task name for this subevent.
taskid	Cisco IOS task ID for this subevent.
is_percent	TRUE or FALSE depending on whether the value is a percentage value.

Subevent Type	Description
value	Total memory use in KB or the actual average memory utilization for this measured interval.
diff	The percentage difference between the oldest sample in the measured interval and the latest sample; a negative value represents a decrease.
sec , **msec**	Elapsed time period for this measured interval.

event_register_ipsla

Registers for an event that is triggered by the **event ipsla**command. Use this Tcl command to publish an event when an IPSLA reaction is triggered. The group ID or the operation ID is required to register the event.

Syntax

```
event_register_ipsla [tag ?] group_name ? operation_id ? [reaction_type ?]
[dest_ip_addr ?][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.
group_name	(Mandatory) Specifies the IP SLAs group name.
operation_id	(Mandatory) Specifies the IP SLA operation ID. Number must be in the range from 1 to 2147483647.

reaction_type	(Optional) Specifies the reaction to be taken for the specified IP SLAs operation. Type of IP SLAs reaction--One of the following keywords can be specified: **connectionLoss**, **icpif**, **jitterAvg**, **jitterDSAvg**, **jitterSDAvg**, **maxOfNegativeDS**, **maxOfNegativeSD**, **maxOfPositiveDS**, **maxOfPositiveSD**, **mos**, **packetLateArrival**, **packetLossDS**, **packetLossSD**, **packetMIA**, **packetOutOfSequence**, **rtt**, **timeout**or **verifyError** can be specified. Type of IP SLAs reaction. One of the following keywords can be specified: • connectionLoss • icpif • jitterAvg • jitterDSAvg • jitterSDAvg • maxOfNegativeDS • maxOfNegativeSD • maxOfPositiveDS • maxOfPositiveSD • mos • packetLateArrival • packetLossDS • packetLossSD • packetMIA • packetOutOfSequence • rtt • timeout • verifyError
dest_ip_address	(Optional) Specifies the destination IP address of the destination port for which the IP SLAs events are monitored.

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 SSSSSSSSS[.MMM] format, where SSSSSSSSS must be an integer representing seconds between 0 and 31536000, 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_reqinfo

"event_ID %u event_type %u event_pub_sec %u event_pub_msec %u event_severity %u" "group_name %u operation_id %u condition %u reaction_type %u dest_ip_addr %u" "threshold_rising %u threshold_falling%u measured_threshold_value %u" "threshold_count1 %u threshold count2 %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	The type of event to monitor for the create, update, and delete flow.
event_type_string	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	The severity of the event.
group_name	The name of theIPSLA group.
operation_id	The IPSLA operation ID.
condition	The condition of IPSLA, which can be one of the following: • cleared • occurred
reaction_type	The IPSLA reaction type.
dest_ip_address	The IPSLA destination IP address.
threshold rising	The IPSLA configured rising threshold value.
threshold falling	The IPSLA configured falling threshold value.
measured_threshold_value	The measured threshold value of the IPSLA operation.
threshold_count1	Corresponds to the argument of the threshold type1.
threshold_count2	Corresponds to the argument of the threshold type2.

event_register_mat

Registers for a MAT event. Use this Tcl command extension to generate an event when a mac-address is learned in the mac-address-table.

Syntax

```
event_register_identity [tag ?] interface ?
[mac-address ?]
[type {add | delete}]
[hold-down ?]
[maxrun ?]
```

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.
interface	A regular expression pattern to match against interface names.
mac-address	Mandatory if the interface parameter is not specified. A regular expression pattern that can be used to filter events by mac addresses of the remote device.
type	(Optional) Filter based on a mac-address-table event type of add or delete. If not specified, the event type is not used in determining whether the event should be triggered.
hold-down	(Optional) When a mac-address-table event comes in, the hold-down timer can be set to make the event to wait between 1 and 4294967295 seconds before processing the policy. If not set then the policy is not delayed in being processed.
maxrun	(Optional) Maximum run time of the script (specified in SSSSSSSSSS[.MMM] format, where SSSSSSSSSS must be an integer representing seconds between 0 and 31536000, 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.

Result String

None

Set _cerrno

No

Event_reqinfo For EEM_EVENT_MAT

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u notification %u intf_name %u mac_address {%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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, at which the event was published to the EEM.
event_severity	The severity of the event.
notification	Notification type--add or delete.
intf_name	The interface name for the address table entry.
mac_address	The mac-address for the address table entry.

event_register_neighbor_discovery

Registers for a neighbor discover event. Use this Tcl command extension to generate an event when a Cisco Discovery Protocol (CDP) or Link Layer Discovery Protocol (LLDP) cache entry or a interface link status changes.

Syntax

```
event_register_neighbor_discovery [tag ?] interface ?
[cdp {add | update | delete | all}]
[lldp {add | update | delete | all}]
[link-event]
[line-event]
[queue_priority {normal | low | 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.
interface	A regular expression pattern to match against interface names.
cdp	Trigger an event when a matching CDP event occurs. One of the following options should be specified. • add--Trigger events only when a new CDP cache entry is created in the CDP table. • all--Trigger an event when a CDP cache entry is added or deleted from the CDP cache table and when a remote CDP device sends a keepalive to update the CDP cache entry. • delete--trigger events only when a CDP cache entry is deleted from the CDP table. • update--trigger an event when a CDP cache entry is added to the CDP table or when the remote CDP device sends a CDP keepalive to update the CDP cache entry.
lldp	Trigger an event when a matching lldp event occurs. One of the following options should be specified. • add--Trigger events only when a new cdp cache entry is created in the cdp table. • all--Trigger an event when a cdp cache entry is added or deleted from the cdp cache table and when a remote cdp device sends a keepalive to update the cdp cache entry. • delete--trigger events only when a cdp cache entry is deleted from the cdp table. • update--trigger an event when a cdp cache entry is added to the cdp table or when the remote cdp device sends a cdp keepalive to update the cdp cache entry.
line-event	Trigger an event when the interface line protocol status changes.
link-event	Trigger an event when the interface link status changes.

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. 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 SSSSSSSSS[.MMM] format, where SSSSSSSSS must be an integer representing seconds between 0 and 31536000, 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_reqinfo For EEM_EVENT_NEIGHBOR_DISCOVERY

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u nd_notification {%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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, at which the event was published to the EEM.
event_severity	The severity of the event.
Common Event_Reqinfo	
nd_notification	The type of notification--cdp-add, cdp-update, cdp-delete, lldp-add, lldp-update, lldp-delete, link, line.
nd_intf_linkstatus	The current interface link status, up or down.
nd_intf_linestatus	The current interface line status, down, goingdown, init, testing, up, reset, admindown, deleted.
nd_local_intf_name	The local interface name for the event.
nd_short_local_intf_name	The short name of the local interface for the event.
nd_port_id	The port id as identified by either the cdp or lldp protocol. This is not set for link or line protocol events.
CDP-specific Event_reqinfo	
nd_protocol	Identifies which protocol triggered the event, for CDP it will always be set to cdp.
nd_proto_notif	Identifies which type of protocol event triggered the event, add, update or delete.
nd_proto_new_entry	If set to 1, the event was triggered because the cache entry is new, otherwise it will be set to 0.
nd_cdp_entry_name	The name of the cdp cache entry in the cdp table.

Event Type	Description
nd_cdp_hold_time	The time remaining until the cdp cache entry expires and is deleted from the cdp table. This time will be reset to some maximum by an update from the cdp neighbor. It is usually set to 0 for new entries.
nd_cdp_mgmt_domain	The CDP VTP management domain.
nd_cdp_platform	The platform name reported by the remote device.
nd_cdp_version	The version of code running on the remote device.
nd_cdp_capabilities_string	The contents of the CDP capabilities field in a string format: Router, Trans-Bridge, Source-Route-Bridge, Switch, Host, IGMP, Repeater, Phone, Remotely-Managed device, CVTA phone port, Two-port Mac Relay or any combination of these separated by commas.
nd_cdp_capabilities_bits	The CDP capabilities bits in a hexadecimal number preceded with 0x.
nd_cdp_capabilities_bit_[0-31]	A series of values that will be set to YES if that bit in the capabilities field is set or NO if it is not set.
LLDP-specific Event_reqinfo	
nd_protocol	Identifies which protocol triggered the event, for LLDP it will always be set to lldp.
nd_proto_notif	Identifies which type of protocol event triggered the event, add, update or delete.
nd_proto_new_entry	If set to 1, the event was triggered because the cache entry is new, otherwise it will be set to 0.
nd_lldp_chassis_id	The chassis id field from the LLDP cache entry.
nd_lldp_system_name	The system name from the LLDP cache entry.
nd_lldp_system_description	The system description field from the LLDP cache entry.
nd_lldp_ttl	The LLDP time to live field from the LLDP cache entry.
nd_lldp_port_description	The port description field from the LLDP cache entry.

Event Type	Description
nd_lldp_system_capabilities_string	The LLDP system capabilities field from the LLDP cache entry. Provided as a string that can contain O, P, B, W, R, T, C, S or any combination of these separated by commas.
nd_lldp_enabled_capabilities_string	The LLDP enabled system capabilities field from the LLDP cache entry. Provided as a string that can contain O, P, B, W, R, T, C, S or any combination of these separated by commas.
nd_lldp_system_capabilities_bits	The LLDP system capabilities bits field from the LLDP cache entry. Provided as a hexadecimal number preceded by 0x.
nd_lldp_enabled_capabilities_bits	The LLDP enabled capabilities bits field from the LLDP cache entry. Provided as a hexadecimal number preceded by 0x.
nd_lldp_capabilities_bits	The LLDP capabilities bits field from the LLDP cache entry. Provided as a hexadecimal number preceded by 0x.
nd_lldp_capabilities_bit_[0-31]	A series of values that will be set to YES if that bit in the capabilities field is set or NO if it is not set.

event_register_nf

Registers for an event when a NetFlow event is triggered by the **event nf**command. Use this Tcl command to publish an event when an NetFlow reaction is triggered..

Syntax

```
event_register_nf [tag ?] monitor_name ? event_type create|update|delete
exit_event_type create|update|delete event1-event4 ? [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.
monitor_name	(Mandatory) The name of the NetFlow monitor.
event_type	(Mandatory) The type of event to monitor for the create, update, and delete flow.

exit_event_type	(Mandatory) The event-type (create, delete, update) at which the event is rearmed to be monitored again.
event1- event4	(Mandatory) Specifies the event and its attributes to monitor. Valid values are **event1**, **event2**, **event3**, and **event4**. The subevent keywords can be used alone, together, or in any combination with each other, but each keyword can be used only once.
maxrun	(Optional) Maximum run time of the script (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). 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.

Subevent Syntax

```
field ? rate_interval ? event1 only entry_value ? entry_op eq|ge|gt|le|lt|wc
[exit_value ?] [exit_op eq|ge|gt|le|lt|wc] [exit_rate_interval ? event1 only]
```

Subevent Arguments

field	(Mandatory) Specifies the cache or field attribute to be monitored. One of the following attributes can be specified: • **counter** {**bytes** | **packets**}--Specifies the counter fields. • **datalink** {**dot1q** | **mac**}--Specifies the datalink (layer2) fields. • **flow** {**direction** | **sampler**}--Specifies the flow identifying fields. • **interface** {**input** | **output**}--Specifies the interface fields. • **ipv4** *field-type*-- Specifies the IPv4 fields. • **ipv6** *field-type*-- IPv6 fields • **routing** *routing-attrribute* -- Specifies the routing attributes. • **timestamp sysuptime** {**first** | **last**}--Specifies the timestamp fields. • **transport** *field-type*-- Specifies the Transport layer fields.
rate_interval	(Mandatory) Specifies the rate interval value in seconds used to calculate the rate. This field is only valid for event1.
entry_value	(Mandatory) Specifies the field or rate value.
entry_op	(Mandatory) Specifies the field operator. The comparison operator valid values are: • **eq** - Equal to • **ge** - Greater than or equal to • **gt** - Greater than • **le** - Less than or equal to • **lt** - Less than • **wc** - Wildcard
exit_value	(Optional) The value at which the event is rearmed to be monitored again.

exit_op	(Optional) The comparison operator used to compare the current event field or rate value with the exit value; if true, event monitoring for this event is reenabled. The comparison operator valid values are: • **eq** - Equal to • **ge** - Greater than or equal to • **gt** - Greater than • **le** - Less than or equal to • **lt** - Less than • **wc** - Wildcard
exit_rate_interval	(Optional) Specifies the exit rate interval value in seconds used to calculate the exit rate value. This field is only valid for event1.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"event_ID %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u monitor_name %u event1-event4_field %u event1-event4_value
```

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	The type of event to monitor for the create, update, and delete flow.
event_type_string	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	The severity of the NetFlow event.

montior_name	The name of the NetFlow monitor.
event1-event4_field	Specifies the event and its attributes to monitor. Valid values are **event1**, **event2**, **event3**, and **event4**.
event1-event4_value	Specifies the event value and its attributes to monitor. Valid values are **event1**, **event2**, **event3**, and **event4**.

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 [tag ?] [sync {yes|no}] [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.
sync	(Optional) A "yes" or a "no" is required to complete this keyword. • If the yes keyword is specified, the policy will run synchronously with the CLI command. • If the no keyword is specified, the policy will run asynchronously with the CLI command.
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.

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 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). 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_reqinfo

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
event_severity %u arg %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	The severity of the event.
argc **arg1** **arg2** **arg3** **arg4** **arg6** **arg7** **arg8** **arg9** **arg10** **arg11** **arg12** **arg13** **arg14** **arg15**	The parameters that are passed from the XML SOAP command to the script.

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 [tag ?] [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.
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.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
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.

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 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 [tag ?] abort|term|start|user_restart|user_shutdown
[sub_system ?] [version ?] [instance ?] [path ?] [node ?]
[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.
abort	(Mandatory) Abnormal process termination. Process may abort 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.
user_restart	(Mandatory) Process termination due to the process restart request from the CLI command.
user_shutdown	(Mandatory) Process termination due to the process kill request from the CLI command.
sub_system	(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.
version	(Optional) Version number of the process assigned by the version manager. Must be of the form major_number.minor_number.level. If specified, each component of the version number must be an integer between 1 and 4294967295, inclusive.
instance	(Optional) Process instance ID. If specified, this argument must be an integer between 1 and 4294967295, inclusive.
path	(Optional) Process pathname (a regular expression string). If the value of the process-name argument contains embedded blanks, enclose it in double quotation marks. Use path ".*" to match all processes.

node	(Optional) 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 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: • 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. If multiple arguments are specified, the process event will be raised when all the conditions are matched.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
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.

Event Type	Description
last_respawn_sec last_respawn_msec	The 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.

event_register_resource

Registers for an Embedded Resource Manager (ERM) event. Use this Tcl command extension to run a policy on the basis of an ERM event report for a specified policy. ERM events are screened by the EEM Resource event detector, allowing an EEM policy to be run when a match occurs for the specified ERM policy.

Syntax

```
event_register_resource policy policy-name [queue_priority low|normal|high|last]
[maxrun ?] [nice 0|1]
```

Arguments

policy	(Mandatory) Specifies the use of a policy.
policy-name	(Mandatory) Name of an ERM policy.

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 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). 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_reqinfo

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

```
"owner_id %lld user_id %lld" time_sent %llu dampen_time %d notify_data_flags %u"
"level {%s} direction {%s} configured_threshold %u current_value %u"
"policy_violation_flag {%s} policy_id %d"
```

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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
owner_id	The Embedded Resource Manager (ERM) owner ID.
user_id	The ERM user ID.
time_sent	The ERM event time, in nanoseconds.
dampen_time	The ERM dampen time, in nanoseconds.
notify_data_flags	The ERM notify data flag.
level	The ERM event level. The four event levels are normal, minor, major, and critical.
direction	The ERM event direction. The event direction can be one of the following: up, down, or no change.
configured_threshold	The configured ERM threshold.
current_value	The current value reported by ERM.
policy_violation_flag	The ERM policy violation flag; either false or true.
policy_id	The ERM policy ID.

event_register_rf

Registers for a Redundancy Facility (RF) event. Use this Tcl command extension to run a policy when an RF progression or status event notification occurs.

Syntax

```
event_register_rf [tag ?] event ?
```

```
[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.
event	(Mandatory) Name of the RF progression or status event. Valid values are: • RF_PROG_ACTIVE • RF_PROG_ACTIVE_DRAIN • RF_PROG_ACTIVE_FAST = 200 • RF_PROG_ACTIVE_PRECONFIG • RF_PROG_ACTIVE_POSTCONFIG • RF_PROG_EXTRALOAD • RF_PROG_HANDBACK • RF_PROG_INITIALIZATION • RF_PROG_PLATFORM_SYNC • RF_PROG_STANDBY_BULK • RF_PROG_STANDBY_COLD • RF_PROG_STANDBY_CONFIG • RF_PROG_STANDBY_FILESYS • RF_PROG_STANDBY_HOT • RF_PROG_STANDBY_OIR_SYNC_DONE • RF_REGISTRATION_STATUS • RF_STATUS_MAINTENANCE_ENABLE • RF_STATUS_MANUAL_SWACT • RF_STATUS_OPER_REDUNDANCY_MODE_CHANGE • RF_STATUS_PEER_COMM • RF_STATUS_PEER_PRESENCE • RF_STATUS_REDUNDANCY_MODE_CHANGE • RF_STATUS_SWACT_INHIBIT

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 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). 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_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event	RF progression or status event notification that caused this event to be published.

event_register_routing

Registers for an event that is triggered by the **event routing** command. These events are handled by the routing event detector to publish an event when route entries change in Routing Information Base (RIB) infrastructure. Use this Tcl command extension to run a routing policy for this script. The network IP address for the route to be monitored must be specified.

Syntax

```
event_register_routing [tag ?] network ? length [ge|le|ne] [type add|remove|modify|all]
[protocol ?] [queue_priority normal|low|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.
network	Specifies the network IP address. The network number can be any valid IP address or prefix.

length	Specifies the length of the network mask in bits. The bit mask can be a number from 0 to 32. • **ge** --(Optional) Specifies the minimum prefix length to be matched. The **ge** keyword represents greater than or equal to operator. • **le** --(Optional) Specifies the maximum prefix length to be matched. The **le** keyword represents the less than or equal to operator. • **ne** --(Optional) Specifies the prefix length not to be matched. The **ne** keyword represents not equal to operator. When **ge**, **le** and **ne** keywords are not configured, an exact match of network length is processed.
type	(Optional) Specifies the desired policy trigger. The type options are **add**, **remove**, **modify**, and **all**. The default is **all**.
protocol	(Optional) Specifies the protocol value for the network being monitored. One of the following protocols can be used: **all**, **bgp**, **connected**, **eigrp**, **isis**, **iso-igrp**, **mobile**, **odr**, **ospf**, **rip**, and **static**. The default is **all**.

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 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). 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_reqinfo

"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u" "event_severity {%s} %u network %u mask %u protocol %u lastgateway %u distance %u" "time_sec %u time_msec %u metric %u lastinterface %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	The severity of the event.
network	The network prefix in IP address format
mask	The network mask in IP address format
protocol	Type of network protocol.
type	Type of event to add, remove or modify.
lastgateway	The last known gateway.
distance	The administrative distance.
time_sec time_msec	Time of event in seconds and milliseconds, when the event was published to the EEM.
metric	Path metric.
lastinterface	The last known interface.

event_register_rpc

Registers for an event that is triggered by the EEM SSH Remote Procedure Call (RPC) command. These events are handled by the RPC event detector that screens for this event. Use this Tcl command extension to run a RPC policy for this script.

Syntax

```
event_register_rpc [queue_priority {normal | low | high | last}] [maxrun <sec.msec>] [nice
 {0 | 1}] [default <sec.msec>]
```

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 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). 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.

default	(Optional) The time period during which the CLI event detector waits for the policy to exit (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). 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.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u
arg %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.

argc **arg0** **arg1** **arg2** **arg3** **arg4** **arg6** **arg7** **arg8** **arg9** **arg10** **arg11** **arg12** **arg13** **arg14**	The parameters that are passed from the XML SOAP command to the script.

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.

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]
```

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.

oid	(Mandatory) OID number of data element in SNMP dot notation (for example, 1.3.6.1.2.1.2.1.0). The types of OIDs allowed are: • COUNTER_TYPE • COUNTER_64_TYPE • GAUGE_TYPE • INTEGER_TYPE • OCTET_PRIM_TYPE • OPAQUE_PRIM_TYPE • TIME_TICKS_TYPE
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-valis 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.

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 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). 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_reqinfo

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"event_severity {%s} oid {%s} val {%s} delta_val {%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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	SNMP event severity, which can be one of the following values: • normal • minor • major
oid	Object ID of data element, in SNMP dot notation.
val	Value of the data element.
delta_val	Delta value between the value of the policies.

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}
[maxrun ?]
[src_ip_address ?]
[dest_ip_address ?]
[queue_priority {normal|low|high|last}]
[maxrun ?]
[nice {0|1}]
[default ?]
```

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

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.
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. The types of OIDs allowed are: • COUNTER_TYPE • COUNTER_64_TYPE • GAUGE_TYPE • INTEGER_TYPE • OCTET_PRIM_TYPE • OPAQUE_PRIM_TYPE • TIME_TICKS_TYPE
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.
maxrun	(Optional) Maximum run time of the script (specified in ssssssss[.mmm] format, where ssssssss must be an integer representing seconds between 0 and 31536000, 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.
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.
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.

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.
default	(Optional) Specifies the time period in seconds during which the snmp notification event detector waits for the policy to exit. The time period is 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.
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.
direction	(Optional) The direction of the incoming or outgoing SNMP trap or inform PDU to filter. The default value is 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.

Result String

None

Set _cerrno

No

Event_reqinfo

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u
event_severity {%s}" "oid {%s} oid_val {%s} src_ip_addr {%s} dest_ip_addr {%s} x_x_x_x_x
(varbinds) {%s} trunc_vb_buf {%s} trap_oid {%s} enterprise_oid {%s} generic_trap %u
specific_trap %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
oid	An user specified object ID.
oid_val	An user specified object ID value.
src_ip_addr	The source IP address of the SNMP protocol data unit (PDU).
dest_ip_addr	The destination IP address of the SNMP PDU.
x_x_x_x_x (varbinds)	The SNMP PDU varbind information.
trap_oid	Indicates the trap OID value.
enterprise_oid	Indicates the enterprise OID value.
generic_trap	Indicates one of a number of generic trap types. There are seven generic trap numbers zero to six.
specific_trap	Indicates one of a number of specific trap codes.

event_register_snmp_object

Registers for a Simple Network Management Protocol (SNMP) object event. Use this Tcl command extension to replace the value when an SNMP with the specified SNMP-object ID (OID) is encountered on a specific interface or address.

Syntax

```
event_register_snmp_object oid ?
type {int|uint|counter|counter64|gauge|ipv4||oid|string}
sync {yes|no}
skip {yes|no}
[istable {yes|no}]
[default ?]
[queue_priority {normal|low|high|last}]
[maxrun ?]
[nice {0|1}]
```

Arguments

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. The types of OIDs allowed are: • COUNTER_TYPE • COUNTER_64_TYPE • GAUGE_TYPE • INTEGER_TYPE • OCTET_PRIM_TYPE • OPAQUE_PRIM_TYPE • TIME_TICKS_TYPE
type	(Mandatory) OID value type.
sync	(Mandatory) A "yes" means that the EEM policy will be notified. If the applet set_exit_status or Tcl return value is 0, then SNMP will handle the request. If the return value is 1, SNMP will use the value provided by the policy for the get request and will not process the set request. A "no" means that EEM will not be notified and SNMP will handle the request. Only one OID can be associated with a synchronous policy. However, multiple synchronous policies can be registered for the same OID.
skip	Mandatory if the sync argument is "no" and should not exist if the sync argument is "yes." If the skip argument is "yes," it means that SNMP will handle the request. If the skip argument is "no," it means that SNMP will act as if the object does not exist.
istable	(Optional) A value of "no" means the OID is scalar object, and "yes" means the OID is table object.

default	(Optional) The time period during which the SNMP Object event detector waits for the policy to exit (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). If the default time period expires before the policy exits, the default action will be executed. The default action is to process the set or get request normally by SNMP subsystem. If this argument is not specified, the default time period is set to 30 seconds.
maxrun	(Optional) Maximum run time of the script (specified in ssssssss[.mmm] format, where ssssssss must be an integer representing seconds between 0 and 31536000, 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.
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.
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_reqinfo

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u
event_severity {%s}" "oid {%s} request {%s} request_type {%s} value %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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
event_severity	The severity of the event.
oid	The ID of the SNMP object in the received get or set request.
request	The get or set request type.
request_type	The type of request (exact or next).
value	For set requests only. The value to set the object to.

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 [tag ?] [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|last]
```

```
[severity_fatal] [severity_critical] [severity_major]
[severity_minor] [severity_warning] [severity_notification]
[severity_normal] [severity_debugging]
[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) 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) A regular expression used to perform syslog message pattern match. This argument is what the policy uses to identify the logged syslog message.
priority	(Optional) The 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: • 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 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). 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.
severity_xxx	(Optional) The event severity to be screened. If this argument is specified, only messages that are at the specified severity level are screened. See the table titled "Severity Level Mapping For Syslog Events" for the severity level mapping for syslog events.

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

Table 27: 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_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_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
msg	The last syslog message that matches the pattern.

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 [tag ?] watchdog|countdown|absolute|cron
[name ?] [cron_entry ?]
[time ?]
[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.
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) 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 the table titled "Time and Date When CRON Events Will Be Triggered."

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" can be 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 the table titled "Special Strings for cron_entry."

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) 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: • 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.

Table 28: 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 below)
day of week	0-7 (0 or 7 is Sun, or names; see the table titled "Special Strings for cron_entry")

Table 29: 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

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
timer_type	Type of the timer. Can be one of the following: • watchdog • countdown • absolute
timer_time_sec timer_time_msec	Time when the timer expired.
timer_remain_sec timer_remain_msec	The remaining time before the next expiration.

See Also

event_register_timer_subscriber

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|last] [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: • 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.

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

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
timer_type	Type of the timer. Can be one of the following: • watchdog • countdown • absolute
timer_time_sec timer_time_msec	Time when the timer expired.
timer_remain_sec timer_remain_msec	The remaining time before the next expiration.

See Also

event_register_timer

event_register_track

Registers for a report event from the Cisco IOS Object Tracking subsystem. Use this Tcl command extension to trigger a policy on the basis of a Cisco IOS Object Tracking subsystem report for a specified object number.

Syntax

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

Arguments

? (represents a number)	(Mandatory) Tracked object number in the range from 1 to 500, inclusive.
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 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). 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_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
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.

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 this 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 [tag ?] [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|last]
[maxrun ?] [nice 0|1]
```

Each argument is position independent.

Note Operator definitions: and (logical and operation), or (logical or operation), andnot (logical and not operation). For example, "sub12_op and" is defined as raise an event when subevent 1 and subevent 2 are true; "sub23_op or" is defined as raise an event when the condition specified in sub12_op is true or subevent 3 is true. The logic can be diagrammed using: if (((sub1 sub12_op sub2) sub23_op sub3) sub34_op sub4) is TRUE, raise event

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.
timewin	(Optional) Time window within which all of the subevents have to occur in order for an event to be generated (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).
sub12_op	(Optional) Combination operator for comparison between subevent 1 and subevent 2.
sub23_op	(Optional) Combination operator for comparison between subevent 1 and 2 and subevent 3.
sub34_op	(Optional) Combination operator for comparison between subevent 1 and 2 and subevent 3 and subevent 4.
sub1	(Optional) Indicates that subevent 1 is specified.
subevent-description	(Optional) Syntax for the subevent.
sub2	(Optional) Indicates that subevent 2 is specified.
sub3	(Optional) Indicates that subevent 3 is specified.
sub4	(Optional) Indicates that subevent 4 is specified.

node	(Optional) The node name to be monitored for deadlock conditions 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 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 the local node on which the registration is done.
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.

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) A regular expression that specifies the process name that you wish to monitor for deadlock conditions. This subevent will ignore the time window even if it is given.

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

Arguments

procname	(Optional) A regular expression that specifies the process name that you wish to monitor for dispatch_manager status.
op	(Optional) Comparison operator used to compare the collected number of events with the specified value; if true, an event will be raised.
val	(Optional) The value with which the number of events that have occurred should be compared.

period	(Optional) The time period for the number of events that have occurred (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 sample is used.

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

Arguments

procname	(Optional) A regular expression that specifies the process name that you wish to monitor for CPU utilization conditions.
op	(Optional) Comparison operator used to compare the collected CPU usage sample percentage with the specified percentage value; if true, an event will be raised.
val	(Optional) The percentage value with which the average CPU usage during the sample period should be compared.
period	(Optional) The time period for averaging the collection of samples (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 sample is used.

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

Arguments

op	(Optional) Comparison operator used to compare the collected total system CPU usage sample percentage with the specified percentage value; if true, an event will be raised.
val	(Optional) The percentage value with which the average CPU usage during the sample period should be compared.

period	(Optional) The time period for averaging the collection of samples (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 sample is used.

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

Arguments

procname	(Optional) A regular expression that specifies the process name that you wish to monitor for memory usage.
op	(Optional) Comparison operator used to compare the collected memory used with the specified value; if true, an event will be raised.
val	(Optional) A percentage or an absolute value 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 with which the measured value should be compared.
is_percent	(Optional) If 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 to be computed. Otherwise, the time period for the collection samples to be averaged (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 sample is used.

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

Arguments

op	(Optional) Comparison operator used to compare the collected available memory with the specified value; if true, an event will be raised.
val	(Optional) A percentage or an absolute value 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 with which the measured value should be compared.
is_percent	(Optional) If 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 to be computed. Otherwise, the time period for the collection samples to be averaged (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). If this argument is not specified, the most recent sample is used.

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

Arguments

op	(Optional) Comparison operator used to compare the collected used memory with the specified value; if true, an event will be raised.
val	(Optional) A percentage or an absolute value 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 with which the measured value should be compared.
is_percent	(Optional) If 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 to be computed. Otherwise, the time period for the collection samples to be averaged (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). 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

Event_reqinfo

```
"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	An ASCII string that represents the name of the event for this event type.
event_pub_sec event_pub_msec	The time, in seconds and milliseconds, when the event was published to the EEM.
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.

Subevent Type	Description
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

Subevent Type	Description
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 fields above 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.
sec msec	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 fields above 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.
sec msec	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.

Subevent Type	Description
sec msec	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 fields above describe the process whose memory usage 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).
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.
Subevent Type	Description

Subevent Type	Description
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.
sec msec	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.

Subevent Type	Description
sec msec	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).
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.

Subevent Type	Description
sec msec	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:

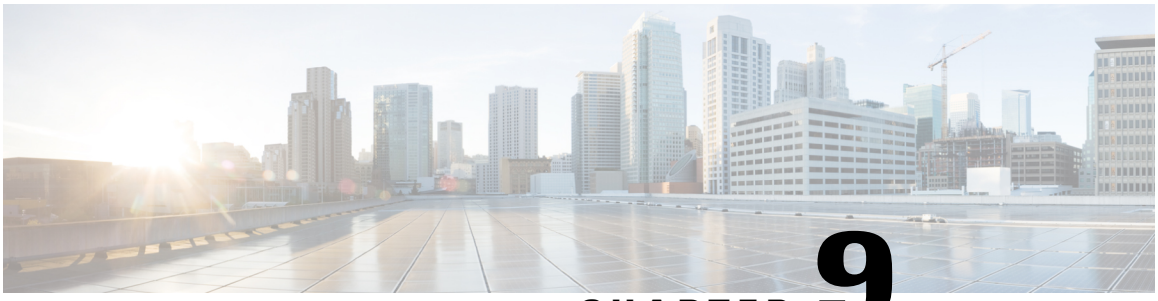
- **used** is 0.
- **avail** is 0.
- **diff** is the percentage difference between the oldest and latest total used memory samples in the specified time window.
- **sec** and **msec** are both the actual time difference between the time stamps of the oldest and latest total used memory samples in this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

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



CHAPTER 9

EEM Event Tcl Command Extensions

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:

priority low|normal|high

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.

- event_completion, page 319
- event_completion_with_wait, page 320
- event_publish, page 321
- event_wait, page 324

event_completion

Sends a notification to the EEM server that the policy is done servicing the event that triggered it. The event only takes a single argument which is the **return_code** of this event instance.

Syntax

```
event_completion status ?
```

Arguments

status	(Mandatory) Exit status (return_code) of this event instance. A value of zero indicates no error and any other integer value indicates an error.

Result String

None

Set _cerrno

No

event_completion_with_wait

The **event_completion_with_wait** command combines the two commands **event_completion** and **event_wait** into a single command for ease of use.

The **event_completion** command sends a notification to the EEM server that the policy is done servicing the event that triggered it. The event only takes a single argument which is the **return_code** of this event instance.

The **event_wait** places the Tcl policy into a sleep state. When the Tcl policy receives a new signal announcing a new event, the policy is placed into a wake state and again returns to a sleep state. This loop continues. If **event_wait** policy is invoked before **event_completed** policy, an error results and the policy exits.

Syntax

```
event_completion_with_wait status ? [refresh_vars]
```

Arguments

status	(Mandatory) exit_status (return_code) of this event instance. A value of zero indicates no error. Any other integer value indicates an error.
refresh_vars	(Optional) Indicates whether built-in and environment variables should be updated (refreshed) from the EEM Policy Director during this event instance.

Result String

None

Set _cerrno

Yes

Sample Usage

Here is a similar example as above using this single command:

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
set i 1
while {1 == 1} {  # Start high performance policy loop
   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 msg "event $i serviced" priority info
   if {$i == 5} {
       action_syslog msg "Exiting after servicing 5 events" priority info
       exit 0
   }
   incr i
   array set _event_state_arr [event_completion_with_wait status 0 refresh_vars 1]
   if {$_event_state_arr(event_state) != 0} {
       action_syslog msg "Exiting: failed event_state " \
                         " $event_state_arr(event_state)" priority info
       exit 0
   }
}
```

Note The running configuration output is the same as the event_publishTcl command.

event_publish

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** 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 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 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 will be 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
}
```

event_wait

Places the Tcl policy into a sleep state. When the Tcl policy receives a new signal announcing a new event, the policy is placed into a wake state and again returns to a sleep state. This loop continues. If **event_wait** policy is invoked before **event_completed** policy, an error results and the policy exits.

Syntax

```
event_wait [refresh_vars]
```

Arguments

refresh_vars	(Optional) Indicates whether built-in and environment variables should be updated (refreshed) from the EEM Policy Director during this event instance.

Result String

None

Set _cerrno

No

Sample Usage

The **event_wait** event detector returns an array type value with a single element named **event_state**. Event_state is a value sent back from the EEM Server indicating whether or not an error has occurred in processing the event. An example of an error here would be if the user configured **event_wait** before configuring **event_completion** when handling the event instance.

The following sample output shows the use of both **event_completion** and **event_wait**Tcl commands:

```
::cisco::eem::event_register_syslog tag e1 occurs 1 pattern CLEAR maxrun 0
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
set i 1
while {1 == 1} {  # Start high performance policy loop
   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 msg "event $i serviced" priority info
   if {$i == 5} {
       action_syslog msg "Exiting after servicing 5 events" priority info
       exit 0
   }
   incr i
   event_completion status 0
   array set _event_state_arr [event_wait refresh_vars 0]
   if {$_event_state_arr(event_state) != 0} {
       action_syslog msg "Exiting: failed event_state " \
                         " $event_state_arr(event_state)" priority info
       exit 0
```

```
    }
}
```

Here is an example of the running configuration:

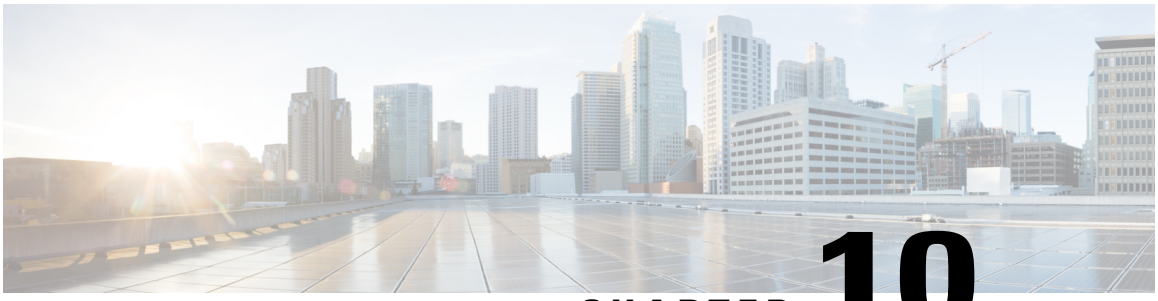
```
Router#
01:00:44: %SYS-5-CONFIG_I: Configured from console by consoleclear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
01:00:49: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:49: %HA_EM-6-LOG: high_perf_example.tcl: event 1 serviced
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
Router#
01:00:53: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:53: %HA_EM-6-LOG: high_perf_example.tcl: event 2 serviced
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
Router#
01:00:56: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:56: %HA_EM-6-LOG: high_perf_example.tcl: event 3 serviced
Router#
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
01:00:59: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Router#
01:00:59: %HA_EM-6-LOG: high_perf_example.tcl: event 4 serviced
01:00:59: %HA_EM-6-LOG: high_perf_example.tcl: Exiting after servicing 5 events
Router#
Router#
Router#copy tftp disk1:
Address or name of remote host [dirt]?
Source filename [user/eem_scripts/high_perf_example.tcl]?
Destination filename [high_perf_example.tcl]?
%Warning:There is a file already existing with this name
Do you want to over write? [confirm]
Accessing tftp://dirt/user/eem_scripts/high_perf_example.tcl...
Loading user/eem_scripts/high_perf_example.tcl from 192.0.2.19 (via FastEthernet0/0): !
[OK - 909 bytes]
909 bytes copied in 0.360 secs (2525 bytes/sec)
Router#
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#no event manager policy high_perf_example.tcl
Router(config)#event manager po high_perf_example.tcl
Router(config)#end
Router#
Router#
Router#
Router#
01:02:19: %SYS-5-CONFIG_I: Configured from console by consoleclear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
01:02:23: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Router#
Router#
01:02:23: %HA_EM-6-LOG: high_perf_example.tcl: event 1 serviced
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
Router#
01:02:26: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:26: %HA_EM-6-LOG: high_perf_example.tcl: event 2 serviced
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
```

```
Router#
01:02:29: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:29: %HA_EM-6-LOG: high_perf_example.tcl: event 3 serviced
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
Router#
01:02:33: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Router#
01:02:33: %HA_EM-6-LOG: high_perf_example.tcl: event 4 serviced
Router#
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
Router#
Router#
01:02:36: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:36: %HA_EM-6-LOG: high_perf_example.tcl: event 5 serviced
01:02:36: %HA_EM-6-LOG: high_perf_example.tcl: Exiting after servicing 5 events
Router#
```

Also while an event has been serviced and is waiting for the next event to come in **show event manager policy active** command will display the following output:

```
Router#show event manager policy active
Key: p - Priority        :L - Low, H - High, N - Normal, Z - Last
     s - Scheduling node :A - Active, S - Standby
default class - 1 script event
 no.  job id      p s status  time of event             event type          name
 1    11          N A wait    Mon Oct20 14:15:24 2008   syslog
high_perf_example.tcl
```

In the above example the status is wait. This indicates that the policy is waiting for the next event to come in.



CHAPTER 10

EEM Library Debug Command Extensions

- cli_debug, page 327
- smtp_debug, page 328

cli_debug

Prints a command-line interface (CLI) debug statement to syslog. This Tcl command extension is used to print a CLI debug statement to syslog if the **debug event manager tcl cli_library** Cisco IOS CLI command is in effect.

Syntax

```
cli_debug spec_string debug_string
```

Arguments

spec_string	(Mandatory) The spec_string argument is used to indicate the type of debug statement.
debug_string	(Mandatory) The debug_string argument is used to indicate the debugging text.

Result String

None

Set _cerrno

No

smtp_debug

Prints a a Simple Mail Transfer Protocol (SMTP) debug statement to syslog. This Tcl command extension prints a SMTP debug statement to syslog if the **debug event manager tcl smtp_library** Cisco IOS command-line interface (CLI) command is in effect.

Syntax

```
smtp_debug spec_string debug_string
```

Arguments

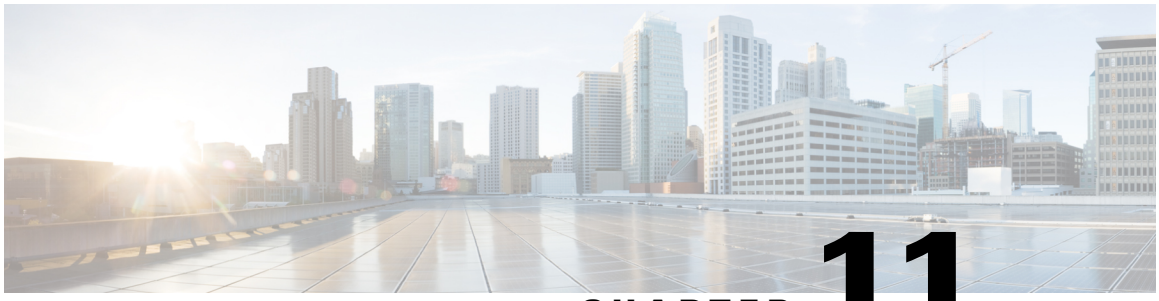
spec_string	(Mandatory) The spec_string argument is used to indicate the type of debug statement.
debug_string	(Mandatory) The debug_string argument is used to indicate the debugging text.

Result String

None

Set _cerrno

No



CHAPTER 11

EEM Multiple Event Support Tcl Command Extensions

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:

priority low|normal|high

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.

- attribute, page 329
- correlate, page 330
- trigger, page 331

attribute

Specifies a complex event.

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

Set _cerrno

No

correlate

Builds a single complex event and allows boolean logic to relate events and tracked objects.

Syntax

```
correlate event ? track ? [andnot | and | or] event ? track ?
```

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.
track	Specifies the event object number for tracking. The range is from 1 to 500. If the tracked object is set, the result of the evaluation is true. If the tracked object is not set or is undefined, the result of the evaluation is false. This result is regardless of the state of the object.

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. **Note** When "and" is used to group events such as traps or syslog messages, then the default trigger occurrence window is three minutes.
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

Set _cerrno

No

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, one or more tracked object states, 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.

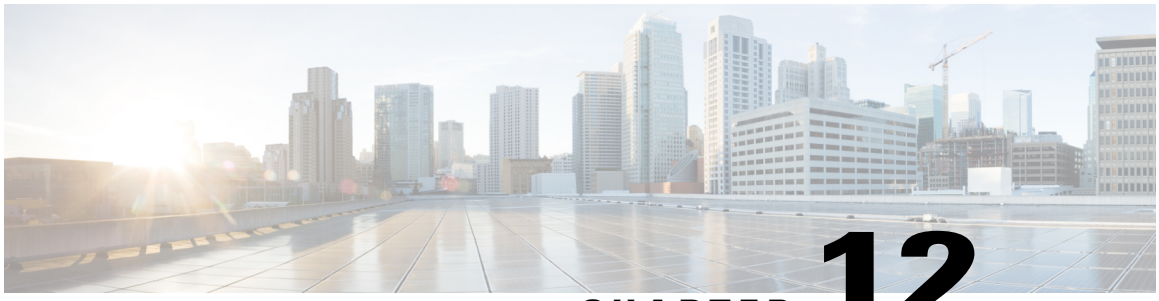
period	(Optional) Time interval in seconds and optional milliseconds, during which the one or more occurrences must take place. This is specified in the format sssssssss[.mmm], where sssssssss must be an integer number representing seconds between 0 and 4294967295, inclusive and mmm represents milliseconds 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 sssssssss[.mmm], where sssssssss 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

Set _cerrno

No



CHAPTER 12

EEM 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**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:

Note Based on RFC 2554, the SMTP e-mail server name--Mailservername-- can be in any one of the following template formats: username:password@host, username@host, or host.

```
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>
Sourceaddr:<space><the IP addresses of the recipients>
Subject:<subject line>
<a blank line>
<body>
```

Note Note that the template normally includes Tcl global variables for configuration.

In a Tcl policy, the port number can be specified by a "Port" line in the e-mail template. If port is not specified, the default port of 25 is used.

Below is a sample e-mail template file:

```
Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc: $_email_cc
Sourceaddr: $_email_ipaddr
Port: <port number>
Subject: From router $routername: Process terminated
```

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


- smtp_send_email, page 334
- smtp_subst, page 335

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) The 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.

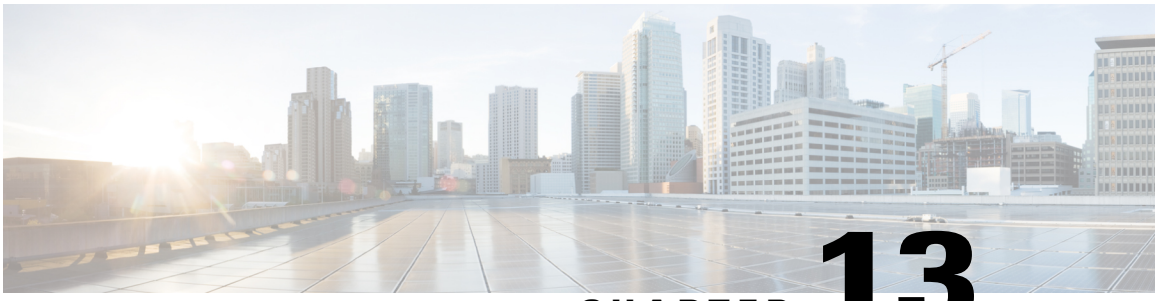
Result String

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

smtp_subst



CHAPTER 13

EEM System Information Tcl Command Extensions

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:

priority low|normal|high

Note All EEM system information commands--**sys_reqinfo_xxx**--have the Set _cerrno section set to yes.

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.

- sys_reqinfo_cli_freq, page 338
- sys_reqinfo_cli_history, page 339
- sys_reqinfo_cpu_all, page 340
- sys_reqinfo_crash_history, page 341
- sys_reqinfo_mem_all, page 342
- sys_reqinfo_proc, page 344

- sys_reqinfo_proc_all, page 345
- sys_reqinfo_routername, page 346
- sys_reqinfo_snmp, page 346
- sys_reqinfo_syslog_freq, page 347
- sys_reqinfo_syslog_history, page 348

sys_reqinfo_cli_freq

Queries the frequency information of all command-line interface (CLI) events.

Syntax

```
sys_reqinfo_cli_freq
```

Arguments

None

Result String

```
rec_list {{CLI frequency string 0},{CLI frequency str 1}, ...}
```

Where each CLI 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}

rec_list	Marks the start of the CLI event frequency list.
time_sec time_msec	Last time when this CLI event was raised.
match count	Number of times that a CLI command matches the pattern specified by this CLI event specification.
raise_count	Number of times that this CLI event was raised. The following fields are information about the CLI event specification: • sync--A "yes" means that event publish should be performed sychronously. The event detector will be notified when the Event Manager Server has completed publishing the event. The Event Manager Server will return a code that indicates whether or not the CLI command should be executed. • skip--A "yes" means that the CLI command should not be executed if the sync flag is not set.

occurs	Number of occurrences before an event is raised; if this argument is not specified, an event is raised on the first occurrence.
period_sec period_msec	Number of occurrences must occur within this number of POSIX timer units in order to raise event; if this argument is not specified, it does not apply.
pattern	Regular expression used to perform CLI command pattern matching.

Set _cerrno

Yes

sys_reqinfo_cli_history

Queries the history of command-line interface (CLI) commands.

Syntax

```
sys_reqinfo_cli_history
```

Arguments

None

Result String

```
rec_list {{CLI history string 0}, {CLI history str 1},...}
```
Where each CLI history string is:

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

rec_list	Marks the start of the CLI command history list.
time_sec time_msec	Time when the CLI command was run.
cmd	Text of the CLI command.

Set _cerrno

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.
sec msec	(Optional) The 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.

sec msec	(Optional) The 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_sec last_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_routername

Queries the device name.

Syntax

```
sys_reqinfo_routername
```

Arguments

None

Result String

```
routername %s
```
Where routername is the name of the device.

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.

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 = 37)    FH_ENOSNMPDATA  (can't retrieve data from SNMP)
```
This error means that there was no data for the SNMP object type.

```
(_cerr_sub_err = 51)    FH_ESTATSTYP  (invalid statistics data type)
```
This error means that the SNMP statistics data type was invalid.

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```
This error means that the event detector was unavailable.

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_sec time_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_sec period_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

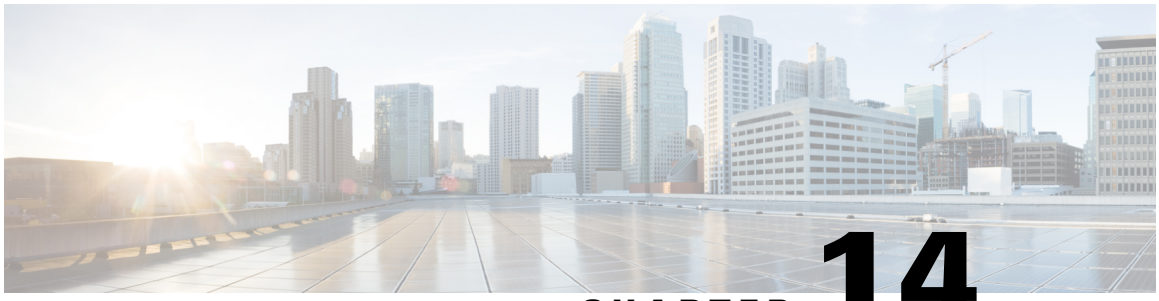
```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```
This error means that the operating system reported an error. The POSIX errno value that is reported with the error should be used to determine the cause of the operating system error.

```
(_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.



CHAPTER 14

EEM Utility Tcl Command Extensions

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:

priority low|normal|high

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.

- appl_read, page 352
- appl_reqinfo, page 353
- appl_setinfo, page 353
- counter_modify, page 354
- description, page 355
- fts_get_stamp, page 356
- register_counter, page 357
- register_timer, page 358
- timer_arm, page 359
- timer_cancel, page 361

- unregister_counter, page 362

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 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 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) The 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 Embedded Event Manager (EEM). This Tcl command extension provides support for saving information in the Embedded Event Manager 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) The string key of the data.
data	(Mandatory) The 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) The counter event ID returned by the **register_counter** Tcl command extension. Must be an integer between 0 and 4294967295, inclusive.
val	(Mandatory) **Note** Mandatory except when the op nop argument value combination is specified. • 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.

description

Provides a brief description of the registered policy.

Syntax

```
description ?
```

Arguments

line	(Optional) Brief description of the policy consisting of 1 to 240 characters.

Result String

None

Set _cerrno

Yes

Sample Usage

The description statement is entered by the author of the policy. It can appear before or after any event registration statement in Tcl. The policy can have only one description.

Note Registration of a policy with more than one description statement will fail.

The following example shows how a brief description is provided for the **event_register_syslog** policy:

```
::cisco::eem::description "This Tcl command looks for the word count in syslog messages."
::cisco::eem::event_register_syslog tag 1 ...
::cisco::eem::event_register_snmp_object tag 2 ...
::cisco::eem::trigger {
       ::cisco::eem::correlate event 1 and event 2
       ::cisco::eem::attribute tag 1 occurs 1
       ::cisco::eem::attribute tag 2 occurs 1
}
```

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) The 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) The 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 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.

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 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) The 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.

INDEX

E

event manager environment command 17