Real-Time Event Detection and Response and Task Automation Using Cisco IOS Embedded Event Manager

Table of Contents

What You Will Learn

This document provides an introduction to Cisco IOS® Embedded Event Manager (EEM) and describes some potential use cases.

Cisco IOS EEM is a robust tool available to customers and is a significant differentiator for Cisco. Cisco IOS EEM is a value-added feature that is included at no cost (on most Cisco® routing and switching platforms) and can reduce customer operating expenses (OpEx) by automating tasks, providing real-time alerts with automated responses, and facilitating troubleshooting. It is essentially a full-time network agent that allows devices to monitor and detect events and immediately take action.

Cisco IOS EEM is embedded in Cisco IOS Software and is enabled by default. There is no requirement or dependency on a centralized management system. Cisco IOS EEM capability is therefore distributed across the network, and Cisco IOS EEM can identify a problem and take action at the point closest to the problem.

This document describes Cisco IOS EEM and provides a technical overview. It also discusses event detectors, policies (using applets and scripts), and use cases.

The appendix provides additional technical details. It discusses multiple event detection, how to gather information using the **event_reqinfo command**, and decision logic. It also presents script action examples, Tool Command Language (TCL) code snippets, and helpful commands and URLs and steps through execution of a sample script.

Note: While this document is based on tests using Cisco Catalyst® 6500 and 4500 Series Switches, the concepts and scripts also apply to other platforms that support Cisco IOS EEM. Cisco IOS EEM Version 2.4 is used as the basis for all discussions in this document, and this version of Cisco IOS EEM is currently supported on the Cisco Catalyst 6500 and 4500 Series using Cisco IOS Software Release 12.2(33)SXI (Cisco Catalyst 6500 Series) and 12.2(50)SG (Cisco Catalyst 4500 Series).

What Is Cisco IOS EEM?

Cisco IOS EEM is a unique subsystem within Cisco IOS Software. Cisco IOS EEM is a powerful and flexible tool that can automate tasks and customize the behavior of Cisco IOS Software and the operation of the device. Using Cisco IOS EEM, customers can create and run scripts directly on a router or switch. The scripts are referred to as Cisco IOS EEM policies and can be programmed using integrated Cisco IOS Software commands (creating an applet) or using the TCL scripting language.

Cisco IOS EEM allows customers to:

- Use the intelligence in Cisco IOS Software to detect specific events and respond or alert network administrators to those events
- Automate routine tasks and create custom commands
- Respond to detected events with immediate action, which may include:
  - Making an automated change to the device configuration
  - Gathering information and sending an email
  - Implementing a custom syslog or Simple Network Management Protocol (SNMP) trap

Although command-line interface (CLI) applets provide a quick and easy interface for Cisco IOS EEM, TCL scripts add a great deal more power and flexibility. This document focuses on TCL scripts and includes a few sample applet configurations for comparison.

Cisco IOS EEM Processing

Cisco IOS EEM processing involves two basic functions:

- Event detection: Configure Cisco IOS EEM to respond on the basis of one of the Cisco IOS Software embedded event detectors using applets (embedded in Cisco IOS Software) or TCL scripts (downloaded to flash memory and loaded into the system). Each event detector can trigger a script on the basis of the specific event type. Most scripts are triggered when an event occurs in Cisco IOS Software: for example, when an interface goes down or a threshold is exceeded. However, some events are triggered by time or manual execution.
- Action policy implementation: Depending on the event being triggered, some action will be performed, such as sending an email message, reconfiguring the switch, or installing an IP route.
  - Before any action is taken, some additional information will probably need to be obtained. The **event_reqinfo** facility, an array that is created in Cisco IOS Software for each event detector, is one source of information. CLI commands embedded in the script or applet are also good sources of additional information.
  - Conditional loop logic such as If-Then-Else and If-Then-While statements can also be used to further focus the decision criteria before an action is performed. For example, after an event detector is triggered, a script or applet could invoke a CLI command to get the time using **show clock**. The result could be parsed to determine whether the event occurred within a specific time frame: IF the event occurred within the relevant time frame, THEN perform action A; ELSE perform action B.

Sample Cisco IOS EEM Applications

Here are some examples of Cisco IOS EEM applications:

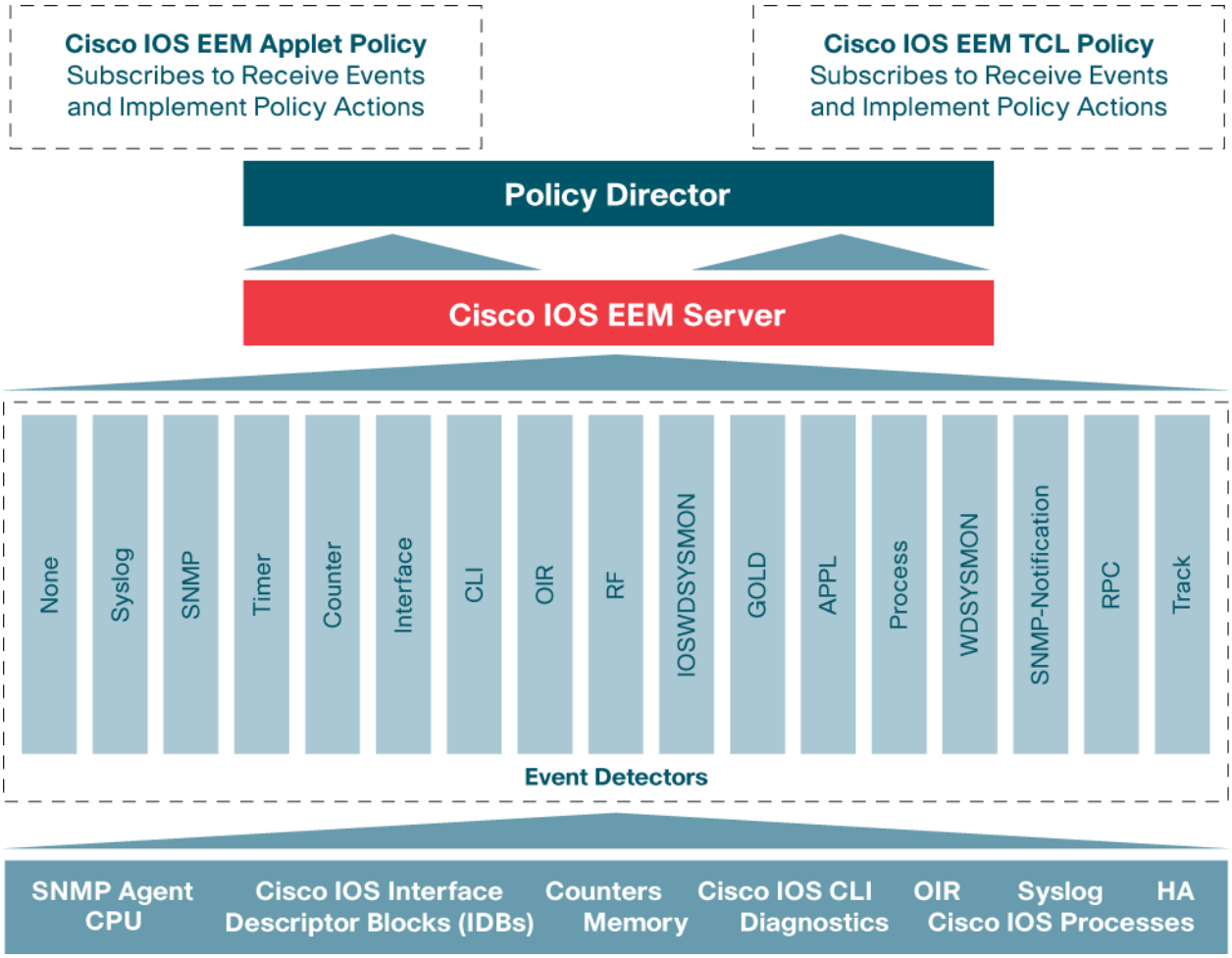
- Automatically configure a switch interface depending on the device connected to the port. For example, if an IP phone is detected, Cisco IOS EEM can automatically configure the port settings for an IP phone.
- Automatically respond to an abnormal condition detected on an interface. For example, if a high error rate is observed on an interface, Cisco IOS EEM can alert network operations or automatically shut down the port and reroute traffic.
- Detect specific syslog messages and collect information. For example, if more information is needed to troubleshoot a sporadic problem, Cisco IOS EEM can gather real-time information upon detection of a specific syslog message. This information can then be automatically emailed to network operations staff for analysis.

The applications for Cisco IOS EEM are limited only by the network administrator's imagination. A Cisco IOS EEM scripting community has been established through which people can share scripts they have found useful. This community is especially valuable for getting some initial ideas about how others have put Cisco IOS EEM to work in their networks. Visit the Cisco Beyond — Product Extension Community at http://www.cisco.com/go/ciscobeyond.

Technical Overview of Cisco IOS EEM

Figure 1 shows the basic Cisco IOS EEM architecture. Event detectors monitor the system for specific events. Events are typically fault conditions, counters, or other occurrences that are deemed interesting. When an event occurs, the event detector passes information to the Cisco IOS EEM server subsystem. If the event matches a registered policy (user-configured applet or script), Cisco IOS EEM performs an action (as specified in the policy).

Figure 1. Basic Cisco IOS EEM Architecture



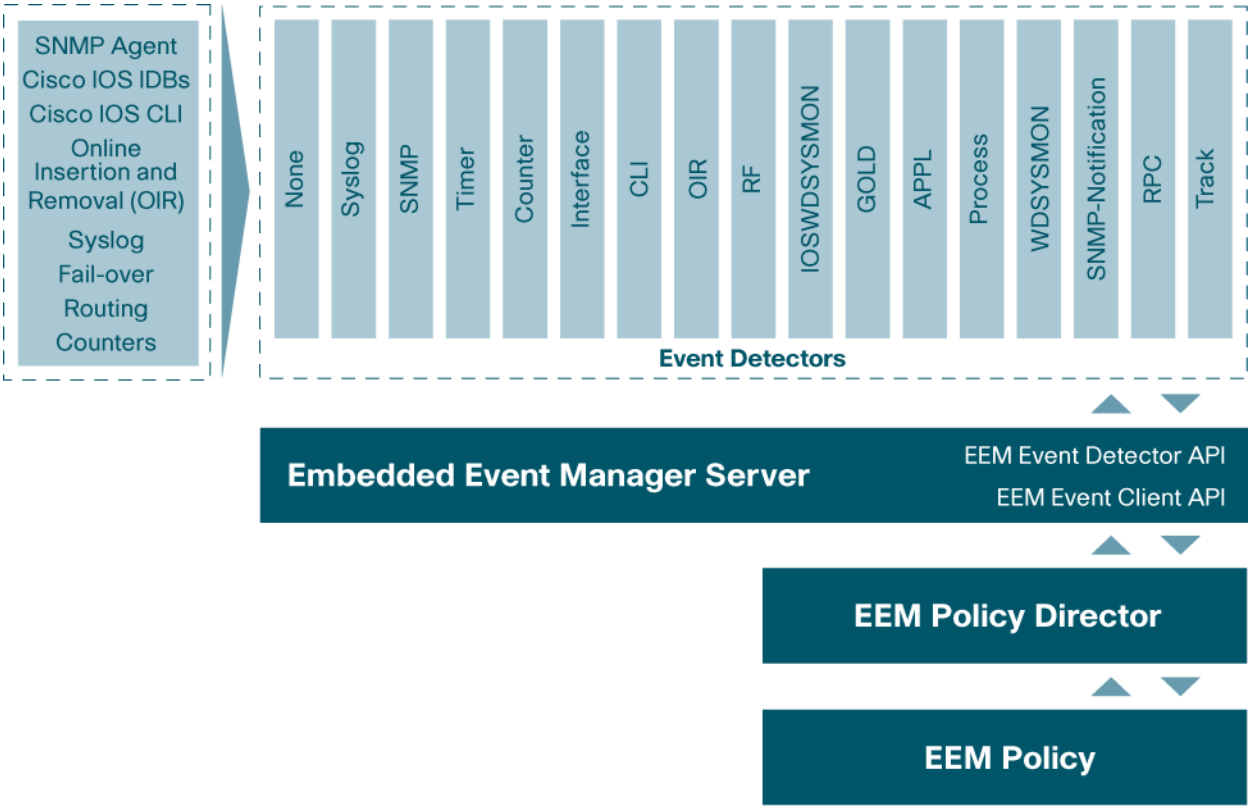
The Cisco IOS EEM server is responsible for the following:

- Event registration: Registration is the starting point for Cisco IOS EEM; an event detector identifies something deemed interesting that happened in the device and provides an update on what happened to the Cisco IOS EEM subsystem.
- Publishing an event: If an event detector detects an event, the Cisco IOS EEM server and the policy director communicate to take action.
- Requesting more information about an event: A script can request more information about an event, which can be used in scripting logic or sent to an administrator.
- Storage and retrieval of persistent data: The Cisco IOS EEM server has facilities to store variables and counters and even to access the CLI to make system changes.
- Registering script directories: A script directory must be defined. This process is described in the section "TCL Implementation: Basic Steps for Registration" later in this document.
- Registering TCL scripts: TCL scripts must be registered with the Cisco IOS EEM server before they become active. This process is also described in the section "TCL Implementation: Basic Steps for Registration" later in this document.
- Registering applets: Applets are registered with the Cisco IOS EEM server in the first line of the applet (written using CLI and stored in the running configuration). Whereas TCL scripts can be unregistered to

prevent execution, applets are stored in the running configuration and must be removed from the configuration to prevent execution.

Figure 2 shows a more detailed view of a Cisco IOS EEM implementation on the Cisco Catalyst 6500 Series. The current implementation is based Cisco IOS EEM Version 2.4, which adds event detectors and programmable actions with the TCL subsystem.

Figure 2. Detailed Cisco IOS EEM Architecture on the Cisco Catalyst 6500 Series

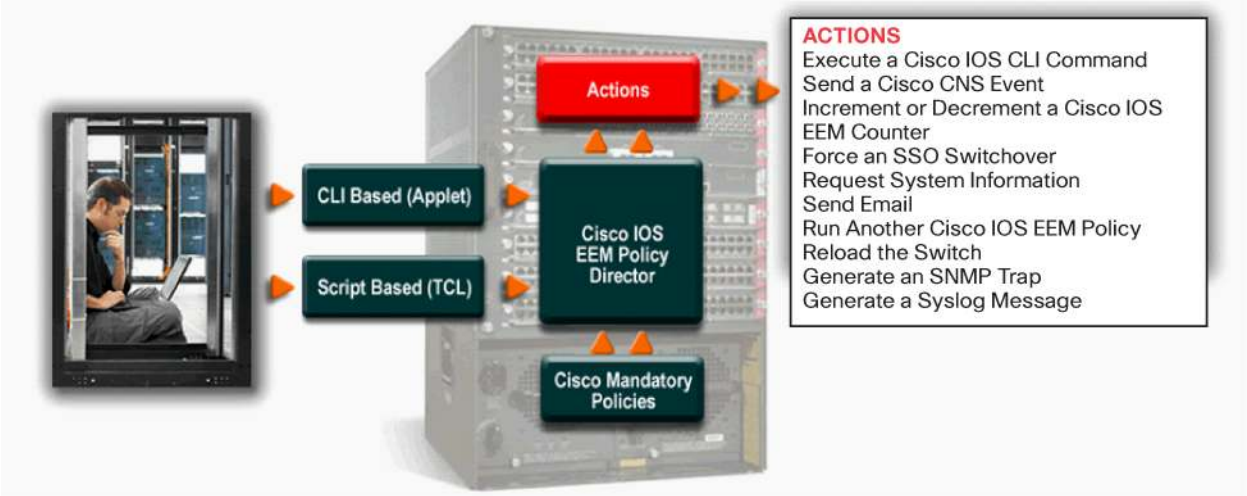


Cisco IOS EEM capabilities will continue to expand with the future addition of more event detectors, integration with additional subsystems and external systems, and expanded platform support. As of Cisco IOS Software Release 12.2(33)SXI, the Cisco Catalyst 6500 Series supports 17 different event detectors. The Cisco Catalyst 4500 Series as of Cisco IOS Software Release 12.2(50)SG supports 15 event detectors (Cisco IOS EEM Version 2.4). The two event detectors not supported by the Cisco Catalyst 4500 Series are the Cisco Generic Online Diagnostics (GOLD) and Enhanced Object Tracking (EOT) event detectors.

A Cisco IOS EEM policy is created through an applet (entered from the CLI) or written externally using a TCL script. The policy is registered with the Cisco IOS EEM server through the Cisco IOS EEM policy director using internal APIs. When an event occurs that matches a registered event specification, the event detector interacts with the Cisco IOS EEM server using internal APIs. Actions are then implemented through the Cisco IOS EEM policy director as shown in Figure 3.

Note: Any Cisco IOS software CLI command (or combination of commands) can be invoked as an action in an applet or script. This powerful capability can be used to alter the running configuration of the device in real time. A script can also collect show command output, which can then be parsed and emailed to provide more information about the event.

Figure 3. Registration of an Cisco IOS EEM Policy and Actions That May Be Implemented



If CLI commands are invoked from within the script, the script opens a telnet (vty) session to access the switch's CLI (note that there is no authentication prompt for the script; instead, the script starts in user execution mode). The script must enter commands as if they were being entered directly at the CLI. For example, **enable** must be specified to enter privileged execution mode, and **configure** terminal must be specified to enter global configuration mode. If the CLI command generates interactive prompts (which may interfere with script execution), the command **file prompt quiet** may help.

Event Detectors

Cisco IOS EEM event detectors are Cisco IOS Software processes that determine when a Cisco IOS EEM event occurs. Each event detector monitors a subset of the operating state of the device. When an event is detected, the event detector communicates with the Cisco IOS EEM server through internal APIs, and immediate action may be taken.

Cisco IOS EEM can monitor many types of events, and these fall broadly into two categories: user and system events. For example, a user event may be triggered when a user types a specific command. A system event may be triggered when a device generates a specific syslog message.

Some event detectors do not monitor Cisco IOS Software. For example, the Timer event detector is used to count down a timer, or run a cron time-based scheduler job periodically, or run a job at a particular date and time. Another example is the Counter event detector, which monitors a generic counter that is build manually by the user.

Each event detector is unique, and the syntax for each varies to identify what the event detector is monitoring. For example, while the Syslog event detector is configured to monitor messages, the Interface event detector is configured to monitor counters provided on interfaces. The GOLD event detector monitors the severity level of Cisco Generic Online Diagnostics (GOLD) results, while the Process event detector looks for restarts or shutdowns in Cisco IOS Software Modularity.

The variety of event detectors provides for a wide range of detectable events in Cisco IOS Software. General-purpose event detectors such as CLI and Syslog provide a means of detecting more universal events, while event detectors such as OIR and GOLD focus on particular processes. It is this diversity that makes Cisco IOS EEM such a powerful tool.

Some event detectors are specific to Cisco IOS Software Modularity code or Cisco IOS Software routing platforms. Updated information regarding Cisco IOS EEM, platform support, and the event detectors available can be found at http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemo.html.

For more information about Cisco IOS EEM support on router platforms (as well as case studies, data sheets, and additional white papers), see http://www.cisco.com/go/eem.

The remainder of this section provides more detail about the event detectors currently available on the Cisco Catalyst 6500 Series (Cisco IOS Software Release 12.2(33)SXI) and Cisco Catalyst 4500 Series (Cisco IOS Software Release 12.2(50)SG).

Event Detector: None

The None event detector publishes an event when the Cisco IOS Software **event manager run** CLI command executes a Cisco IOS EEM policy. Cisco IOS EEM schedules and runs policies on the basis of an event specification that is contained within the policy itself. Before the **event manager run** command will run, the Cisco IOS EEM policy must be identified and registered with the system to be permitted to run manually.

Unlike all the other event detectors, the user triggers the None event detector manually. This event detector can be used as a macro to enter CLI commands or to test the logic of a script or applet. It can also be used to obtain information through, for example, CLI commands and apply some logic to determine what actions need to be performed. The main point to remember is that this event detector is not triggered by any Cisco IOS Software event; instead, it is triggered manually by the user.

Event Detector: Syslog

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

For example, Cisco IOS EEM can be configured to screen the syslog for an interface-down message by looking for the string **%LINK-3-UPDOWN** in the syslog message. A regular-expression pattern match could be applied to the entire syslog message to find the interface name. Then a syslog message can be sent to the console, or an email message can be sent to the on-call engineer.

This event detector is fairly universal in that it tracks any event that sends a syslog message. As new features are added, this event detector provides a quick way to gain management visibility prior to availability of SNMP MIB support.

Event Detector: 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 value matches user-specified values or crosses specified thresholds.

A threshold can be specified as an absolute value, average rate of change, or incremental value. If the incremental value is set to 50, for example, an event would be published when the interface counter increases by 50. The values are obtained for comparison to thresholds after a user-configured poll interval. After an SNMP event has been published, the monitoring logic is reset using either of two methods: the interface counter is reset when a second threshold, called an exit value, is crossed, or when a set period of time has elapsed.

If SNMP has traps or SNMP GET commands that specify that an SNMP manager be notified when a threshold is crossed, Cisco IOS EEM extends this notification by allowing users to send email or post a syslog message to the console. In addition, a policy action can be performed such as reconfiguration of the switch to mitigate the effects of crossing the threshold.

Event Detector: Timer

The Timer event detector publishes events for the following four 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.

Similar to the None event detector, the Timer event detector is not triggered by a Cisco IOS Software process such as a Cisco GOLD or OIR process, but by the user, who configures some sort of timer to trigger an event; essentially, time initiates the event. When the timer is triggered, most likely some information will be obtained, and a policy will be applied dependent on the information received.

For example, a cron timer can be used to run a particular command at the same time everyday, or a watchdog timer can run the same command repeatedly after a configured amount a time.

The Timer event detector also can be configured to perform a **show ip route** command looking for a particular route. If the route is not in the routing table, this could indicate that packets are traversing a backup link, which may be more costly. The switch can then notify the on-call engineer that the switch is using the more costly route. Checking this route periodically, even just daily, can mitigate the cost incurred by using the backup link.

Event Detector: Counter

The Counter event detector publishes an event when a named counter crosses a specified threshold. Multiple participants affect counter processing. One or more subscribers define the criteria that cause the event to be published, and the event detector can modify the counter. 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.

The counter itself that the Counter event detector monitors needs to be created and needs to incremented or decremented. Most likely, at least one other script will be written whose job is to create a counter for the Counter event detector to monitor.

For example, consider a use case in which the goal is to send a notification if an interface goes down five times in 3 minutes. The Syslog event detector can be used to monitor the interface and increment a counter whenever the interface goes down. This counter can then be monitored by a Counter event detector. To limit the counter to 3-minute intervals, a Timer event detector can reset the counter every 3 minutes. If the counter reaches 5 within the 3-minute period, the Counter event detector can send an email notification that the interface is flapping. As an additional action, the script can also reset the counter.

Event Detector: Interface

The Interface event detector publishes an event when an 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 were 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 either of two methods. The interface counter is reset when a second threshold, called an exit value, is crossed, or when an period of time elapses. When configuring this event detector, the user must indicate which interface and which parameter to monitor. The user must also configure the polling interval to indicate how often the reading is taken.

Consider the previous example using the Counter event detector. The same result could be obtained by using the Interface event detector. The Interface event detector could be configured to monitor interface resets for a particular interface every 3 minutes, looking for an incremental value of 5 during the 3-minute polling interval. When triggered, the Interface event detector could send the same email notification indicating a flapping interface.

Event Detector: CLI

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 Cisco IOS EEM policy exits, and the Cisco IOS EEM policy can control whether the command is executed. A read-and-write variable called **_exit_status**, can be used with this detector; it allows you to set the exit status at the policy exit for policies triggered by synchronous events. If, for example, the variable **_exit_status** is set to **0**, the CLI command invoked by the user is skipped, whereas, if the variable **_exit_status** is set to **1**, the command is run.
- Asynchronous publishing of CLI events: The CLI event is published, which invokes the script, and then the CLI command that the user entered (invoking this script) is executed.
- Asynchronous publishing of CLI events with command skipping: The CLI event is published, but the CLI command entered by the user is not executed.

The synchronous parameter cited in the preceding list enables the system to hold the command while the script or applet gathers more information to decide whether or not to allow the command to be executed. In the example presented in this document, the use case prevents debug commands (with the exception of **show debugging** and **undebug all**) from being run during business hours. This example illustrates the use of the synchronous parameter to hold the execution of the command while the script determines the type of command that has been entered (such as **show debugging** or **undebug all**).

Note that the system matches on the entire command and not on abbreviated input. For example, the system will see a **sh debug** entered by the user as **show debugging**. When building a pattern match, you must use the full command and not the abbreviated version. If you are uncertain as to what the full command is, use the tab function in Cisco IOS Software to extend each word in a command.

Event Detector: OIR

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

- A line card is removed from the chassis.
- A line card is inserted into the chassis.

Route processors, line cards, and feature cards can be monitored for OIR events. With knowledge of the insertion of a module, the module can be automatically configured according to the line-card type. When the module is inserted, the module type can be obtained through a **show** command. Depending on the type of card that was inserted, the applet or TCL script can configure the module with the desired configuration, such as security (port security, Dynamic Address Resolution Protocol [ARP] Inspection, Dynamic Host Configuration Protocol [DHCP] snooping, etc.) or spanning-tree host mode (PortFast enabled).

Event Detector: RF

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

Event Detector: IOSWDSYSMON

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

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

Two events can 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.

As with other threshold event detectors, a subevent or process to be monitored is configured in IOSWDSYSMON. You can use general operands — greater than, greater than or equal to, etc. — to compare detected values to the value desired. Other parameters specify monitoring of two events at the same time. You can specify the event publishing criteria to require one or both events to cross their specified thresholds before the event detector is triggered. In addition, you can specify a time window in which the subevents must occur for the event detector to be triggered.

While a similar result could be obtained using the SNMP event detector monitoring CPU object IDs (OIDs), this event detector is specifically for monitoring CPU and memory.

Event Detector: GOLD

Cisco Generic Online Diagnostics (GOLD) on Cisco Catalyst 6500 Series Switches checks the health of hardware components and verifies proper operation of the system data and control planes. These tests take effect when the system is booting and when the system is operational.

Boot diagnostics allow the system to detect faults in the hardware components at boot time and help ensure that a failing module is not introduced into a live network. If boot diagnostics detect a failure on a Cisco Catalyst 6500 Series Switch, the failing modules are shut down. The administrator can configure the level of boot diagnostics as minimal, complete, or disabled. Though use of complete diagnostics is recommended, the default on the Cisco Catalyst 6500 Series is minimal diagnostics, allowing the system to come online faster.

Run-time diagnostics run a series of diagnostic checks to determine the condition of an online system. Diagnostic tests can be disruptive and nondisruptive. Nondisruptive tests occur in the background and do not affect the system data or control planes, but disruptive tests do affect live packet flows and should be scheduled during special maintenance windows.

The GOLD event detector is triggered when Cisco GOLD boot or run-time diagnostics on a specified card and subcard results in a configured severity level (normal, minor, or major). The GOLD event detector can also be configured to generate an event only if it is a new failure or if it is an existing one. When the event detector is triggered, the applet or TCL script is normally configured to perform an action based on the failure detected.

Event Detector: APPL

The application-specific (APPL) event detector allows any Cisco IOS EEM policy (TCL script or applet) to publish an event. This event can be used to trigger a second script (TCL script or applet) into action. When a Cisco IOS EEM policy publishes an event, it must use a Cisco IOS EEM subsystem number as part of its event creation; specifically, the TCL script or applet must always use subsystem number 798, which is reserved by the Cisco IOS EEM subsystem for this purpose. Along with the event type, the event type number (from 1 to 999) must be specified. This unique number identifies one of several scripts that can be triggered into action. If an existing policy is registered for subsystem 798 and for the specified event type number, a second policy of the same event type will trigger the first policy to run when the specified event is published.

Event Detector: Process

Unlike Cisco IOS Software, Cisco IOS Software Modularity on the Cisco Catalyst 6500 Series can run processes independently. With this capability comes to capability to restart and shut down processes without affecting other processes. The Process event detector generates events for Cisco IOS Software Modularity process start, normal and abnormal stop, and restart events. The events generated by the system manager allow policies to change the default behavior of the process restart.

The Process event detector can monitor a single process or many processes at once. If you want to monitor only a particular process, this process can be called out individually by instance or path. If a particular process (path or instance) is not called out in TCL script, the event detector monitors all processes. This approach differs from that of an applet, in which a regular expression is used to identify which process to monitor. In this case, a period (.) denotes any match. Monitoring can also be restricted to a certain CPU using the **node** option. In addition to choosing a process, the type of event can be identified: as an abort event, normal termination, start of a process, user-initiated restart, or user-initiated shutdown.

Event Detector: WDSYSMON

The Cisco IOS Software Modularity watchdog system monitor (WDSYSMON) event detector detects infinite loops, deadlocks, and memory leaks in Cisco IOS Software Modularity processes (Cisco Catalyst 6500 Series only).

Use the WDSYSMON event detector to register a composite event, which is a combination of several subevents or conditions. For example, you can use this command to register the combination of conditions in which the CPU utilization of a certain process exceeds 80 percent and the memory used by the process is greater than 50 percent of its initial allocation.

As with other threshold-type event detectors, a subevent or process is configured in WDSYSMON to be monitored. You can use general operands—greater than, greater than or equal to, etc.—to compare the detected values to the value desired. Other parameters let you monitor up to four events at the same time. You can specify criteria to require any combination of events to cross their thresholds before the event detector is triggered. In addition, you can specify a time window in which the subevents need to occur for the event detector to be triggered.

Event Detector: SNMP-Notification

The SNMP-Notification event detector can be made to run a policy when an SNMP trap with the specified SNMP OID is encountered on a specific interface or address.

The SNMP-Notification event detector can intercept SNMP trap and inform messages coming into the router. An SNMP notification event is generated when an incoming SNMP trap or inform message matches specified values or crosses specified thresholds.

Both SNMP and the SNMP server manager must be configured and enabled on the device prior to the use of the SNMP-Notification event detector.

Event Detector: RPC

The remote procedure call (RPC) event detector provides the capability to invoke Cisco IOS EEM policies from outside the router 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 Cisco IOS EEM policies and then receive output in a SOAP XML-formatted reply.

Event Detector: Track

The enhanced object tracking (EOT) Track event detector publishes an event when the status of a tracked object changes. Object tracking was first introduced in the Hot Standby Router Protocol (HSRP) as a simple mechanism that allowed users to track the interface line-protocol state only. If the line-protocol state of the interface was down, the HSRP priority of the router was reduced, allowing another HSRP router with a higher priority to become active.

EOT is now integrated into Cisco IOS EEM to allow Cisco IOS EEM to report on a status change of a tracked object and to allow enhanced object tracking to track Cisco IOS 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.

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, Virtual Router Redundancy Protocol (VRRP), and Gateway Load Balancing Protocol (GLBP) can register their interest with the tracking process, track the same object, and each take different action when the object changes. A unique number will identify each tracked object. This unique number is specified in the tracking 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.

Another source for object tracking is Cisco IOS IP Service Level Agreements (SLAs). EOT can track a Cisco IOS IP SLA for state, such as Internet Control Message Protocol (ICMP) Echo or TCP Connect. When connectivity is lost, EOT will show the state as down. The Tracking event detector will then recognize the down state and can invoke a policy action. In the use case provided in this document, the Cisco IOS IP SLA is used to verify the reachability of a server IP address. When reachability is broken, the Tracking event detector will recognize this state, and EOT is triggered. The action policy removes a host route to that server saying that it is unavailable. A message is sent to the console, and an email message is sent to the network operations on-call team.

Policies: Applets and Scripts

A Cisco IOS EEM policy is an entity that defines an event and the actions to be taken when that event occurs. There are two types of Cisco IOS EEM policies: applets and TCL scripts. An applet is a simple form of policy that is defined using the CLI. A TCL script is a form of policy that is written in TCL.

Applets

Applets are created through the CLI and become part of the Cisco IOS Software configuration file (and persist across system reboots).

In applet configuration mode, three types of configuration statements are supported.

- An **event** command specifies the event criteria to trigger the applet to run.
- An **action** command specifies an action to perform when the Cisco IOS EEM applet is triggered.
- A **set** command sets the value of a Cisco IOS EEM applet variable.

As of Cisco IOS Software Release 12.2(33)SXI and 12.2(50)SG, up to six **event** commands are allowed in an applet configuration. See the appendix for more information about multiple event detection. Multiple **action** configuration commands also are allowed in an applet configuration. 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.

Explore the following example to gain more insight into the structure of an applet:

```
event manager applet backup-config

event Syslog pattern “%SYS-5-CONFIG”

action 1.0 cli command “enable”
action 2.0 cli command “config t”
action 3.0 cli command “file prompt quiet”
action 4.0 cli command “end”
action 5.0 cli command “copy run disk0:running-config”
action 5.5 cli command “wr t”
action 6.0 cli command “config t”
action 6.2 cli command “no file prompt quiet”
action 6.2.2 cli command “end”
```

The first line,

```
event manager applet backup-config
```

registers the applet with Cisco IOS EEM and enters applet configuration mode.

The second line,

```
event Syslog pattern “%SYS-5-CONFIG”
```

specifies the event criteria that cause the applet to run. In this example, a Cisco IOS EEM event is triggered when the syslog pattern %SYS-5-CONFIG is detected.

The remaining lines specify the action to be taken when the applet is triggered. The argument following the **action** command *(label)* can be any string value, but actions are sorted and run in alphanumeric key sequence. Note in the following code that the statement starting with "4.5" has been added to the applet shown previously and will be inserted between statements 4.0 and 5.0. Likewise, the following code shows a statement starting with "6.2.2" inserted be between statements 6.2 and 7.0.

```
action 1.0 cli command "enable"
action 2.0 cli command "config t"
action 3.0 cli command "file prompt quiet"
action 4.0 cli command "end"
action 4.5 cli command "copy run disk0:running-config"
action 5.0 cli command "wr t"
action 6.2 cli command "config t"
action 6.2.2 cli command "no file prompt quiet"
action 7.0 cli command "end"
```

The preceding example shows a series of CLI commands that enter privileged execution mode and then global configuration mode to add the **"file prompt quiet"** command to the running configuration. This command is useful for applet execution when the CLI generates an interactive prompt requesting some input from the user. This command allows the script to bypass this prompt, which would otherwise require a response to the request. Global configuration mode is then exited, and the running configuration is copied to disk0 and written to NVRAM. Global configuration mode is then again entered to remove the **"file prompt quiet"** command from the running configuration.

Various action types are available within the body of applets and TCL scripts, including the capability to send an email. When an email message needs to be sent within the body of a script, the action syntax for sending the email message is as follows:

action *label* **mail server** *email-server-address* **to** *to-email-address* **from** *from-email-address* [**cc** *cc-address*] **subject** *email-subject* **body** *email-body-text*

An example of this applet statement is shown in the following example:

```
6500(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"
```

For more information about the command structures supported in an applet, go to http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemc.html.

TCL Scripts

TCL is a string-based command language that is interpreted at run time. Cisco IOS EEM supports TCL Version 8.3.4 and adds support for Cisco device-specific extensions using additional TCL libraries. Scripts are written using an ASCII editor on another device, not on the networking device. The script is then copied to the networking device and registered with Cisco IOS EEM. As an enforced rule, Cisco IOS EEM policies are short-lived run-time routines that must be interpreted and executed in less than 20 seconds. If more than 20 seconds is required, the **maxrun** parameter can be specified in the body of the TCL script in the **event_register** statement to specify the amount of run time required by that script.

Cisco IOS EEM policies use the full range of TCL's capabilities. However, Cisco provides enhancements to TCL in the form of TCL command extensions that enhance the ability of the Cisco IOS EEM policies to invoke specific device actions. The main categories of TCL command extensions identify the detected event, the subsequent action, utility information, counter values, and system information.

Cisco IOS EEM allows you to write and implement your own policies using TCL. Writing a Cisco IOS 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 detection of the event
- Choosing the actions to implement the response to the detected event

Using TCL with Cisco IOS EEM

In general, TCL scripts are used with Cisco IOS EEM to accomplish the following:

- Trigger an event: Configure an event detector to trigger when a certain event occurs.
- Perform an action: Perform some action based on the event detector that was triggered.
- Decision criteria: Additional tasks may need to be performed to determine what action to take. To perform additional actions, the following additional steps may be necessary:
  - Collect more information: More information can be obtained by executing CLI commands or by using the **event_reqinfo** command (to see an array that the system creates for each event detector type).
  - Use logic: Conditional loops such as If-Then-Else and If-Then-While can be used determine which action to perform.

As with applets, a TCL script is activated when a particular event is detected by an event detector. The TCL script can then gather additional information about the event. Gathering additional information about the event is achieved using the **event_reqinfo** command. This facility will populate an array with event details, and these details can then be called or referenced from within the script at a later point in the execution path.

You can also apply logic, which can use conditional loop constructs to define a condition before execution of a particular task. The action, which the TCL script can execute, can be any valid CLI command (or combination of commands) or other allowed Cisco IOS EEM action (see Figure 3 earlier in this document).

TCL scripts can use local variables and environment variables. The following section provides more information about the differences between these variable types and the use of variables in a TCL script.

TCL Script Variables

Variables can be used within a script to store information to reference later in the script or for use in another script.

Local variables are created within the script and can be referenced later in the script. These variables only ever have relevance within the body of the executing script. The set command assigns a value to a variable. For example:

```
set mesg “This is a test”
```

The preceding example creates the local variable **mesg** and stores the text “This is a test” in it. The variable can then be used later in the script as shown in the following example:

```
action_Syslog msg “The mesg variable contains: $mesg”
```

This command outputs a syslog message that would read “The mesg variable contains: This is a test.” A $ character must be prepended to variable names when calling them to reference their information.

Environment variables are defined outside of the body of the script and can also be used by multiple scripts that run on the switch. They are useful if you need to define global variables for use by multiple scripts. For example, if multiple scripts send email to an administrator as a result of an event, you may want to define the **_email_server** variable globally as follows (in device configuration mode):

```
event manager environment _email_server smtp.domain.com
```

This statement is entered through the CLI and creates a Cisco IOS EEM environment variable that can be called from multiple scripts (references to **$_email_server** retrieve the value of smtp.domain.com). If a script uses environment variables, typically notes will appear at the beginning of the script to identify the environment variables that are used or required.

Note: The values of environment variables are copied to a script’s local memory prior to script execution. Although environment variables can be changed in a script (through CLI commands), local references to an environment variable always contain the initial copied information. However, if a secondary script is executed within a primary script, the secondary script may copy updated environment variable information prior to execution. An example of this behavior is shown earlier in this document in the use case example for the Interface event detector.

Implementing TCL: Basic Steps for Registration

For a script to run on a switch, it must first be copied onto the device and then registered on the switch.

To load and register TCL scripts on a device, register the script directory (you need to perform this action just once):

1. Register a script directory.
   - The directory must be in bootflash memory or on a removable compact flash drive (typically accessed through disk0: or disk1:).
   - You should create a subdirectory to hold Cisco IOS EEM scripts.
   - Register the directory using this command:

```
event manager directory user policy disk0:/eem
```

After a script directory is defined, perform the following steps (for every script):

1. Copy the script to the registered directory.
   - The script must be created externally using a text editor.
   - The script name must end with file extension .tcl.
   - Use Trivial File Transfer Protocol (TFTP) to send the script to the directory, using this command:

```
copy tftp://<tftp-ip>/<script-name> disk0:/eem
```

2. Register the script with Cisco IOS EEM.
   - Scripts must reside in the registered directory.
   - Register the script using this global configuration command:

```
event manager policy <script-name>
```

After the script is registered, it is loaded into memory and is active (unless you are using the None event detector). To view registered scripts on the system, use this command:

```
show event manager policy registered
```

Note: After a script is registered with Cisco IOS EEM, it is active and waiting for event detection. The only exception is a script that uses the None event detector. None event detector scripts must be run manually from the CLI using this command:

```
event manager run <script-name>
```

If a script is modified and the script file is overwritten, the active script resident in running memory is not automatically updated with the modified information. To activate the modified script, you must unregister the existing (active) script and then reregister the script. To unregister a script, you must enter global configuration mode (**config t**) and use this command:

```
no event manager policy <script-name>
```

After the modified script is copied to the registered directory, you can reregister the script using this command:

```
event manager policy <script-name>
```

The use case for the None event detector in the “Use Cases” section later in this document automates the process of copying a modified script using TFTP, unregistration of the existing script, and reregistration of the modified script.

TCL Script Components

In general, a TCL script includes the following elements:

- Event trigger: Event detector
- Import of Cisco TCL libraries
- Information collection (using **event_reqinfo** and CLI commands; optional)
- Logic (such as If-Then-Else; optional)
- Action

Event Trigger: Event Detector

An event detector is required in every TCL script and defines the event that triggers the script execution. The event detector identifies the type of detection Cisco IOS EEM will use. For example, a CLI event detector will examine events when a user types in the CLI, and the Syslog event detector will examine syslog messages. The GOLD event detector looks at Cisco GOLD diagnostics and can act when a major, minor, or normal event occurs.

The following example shows the syntax for an event detector:

```
::cisco::eem::event_register_cli sync yes pattern “debug” maxrun 20
```

Import of Cisco TCL Libraries

The namespace import command imports TCL libraries that include Cisco command extensions not available in the base TCL command set.

The following TCL syntax must be included in every TCL script that will run as a Cisco IOS EEM script:

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

Information Collection

After a script has been triggered into action, you will likely want to gather more information about the event or capture some command output. The **event_reqinfo** command can provide details about the event that triggered the script. The information provided depends on the event detector used. The following TCL script command will populate an array (called **arr_einfo**) with the event detail information:

```
array set arr_einfo [event_reqinfo]
```

See the appendix for more information about **event_reqinfo**.

For example, if the OIR event detector is used, the array is populated with a series of data fields that provide more information about the event. Specific parameters for each event are created and primed with data about that event. Further details showing an example of this is detailed in the appendix at the end of the document. The information is shown in Table 1.

The detail information can then be referenced by event type. The following commands set variables, which contain the detail information:

```
set module_slot $arr_einfo(slot)
set insertion_type $arr_einfo(event)
set event_type $arr_einfo(event_type)
```

Alternatively, the information can be referenced directly in the script using **$arr_einfo(event type**).

Information can also be gathered through CLI commands. For example, if OIR event detection is triggered, you may want to get a current inventory of the modules installed in the chassis, using the **show module** command.

To invoke a CLI command from a script, add the following code snippet to the script:

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

(Do not worry about the detail of each line.)

This code instructs Cisco IOS EEM to open a telnet session to the device (in user execution mode). The script can then execute CLI commands through this telnet session by adding the following code snippet:

```
    if [catch {cli_exec $cli1(fd) “enable”} result] {
        error $result $errorInfo
    }
```

In the preceding example, the actual CLI command you want to invoke is shown in quotation marks. The command example invokes the **enable** command, which places the CLI in privileged execution mode. You can now add a **show module** command:

```
    if [catch {cli_exec $cli1(fd) “show module”} result] {
        error $result $errorInfo
    } else {
             set cmd_output $result
    }
```

The **show** output of the CLI command is placed in the variable named **result**, which can be seen as the statement following the command in the preceding example. A new variable named **cmd_output** is created and populated with the command output (it copies the contents of the variable **result**). The cmd_**output** variable can now be used later in the script, perhaps, for example, within the body of an email message.

If the CLI was opened in a script, you should always be sure to close the telnet session to free vty resources on the device (vty is a virtual terminal and represents the telnet session). The code snippet to accomplish this is shown here:

```
#-----Close CLI
   cli_close $cli(fd) $cli(tty_id)
```

Logic and Action

After information has been collected, you may want to apply logic in a loop construct (If-Then-Else, While, or For). Action can then be taken, which may include the CLI (changes to the device running configuration), an email alert, a custom syslog message, or a custom trap.

Custom syslog messages are simple to create. Here is an example:

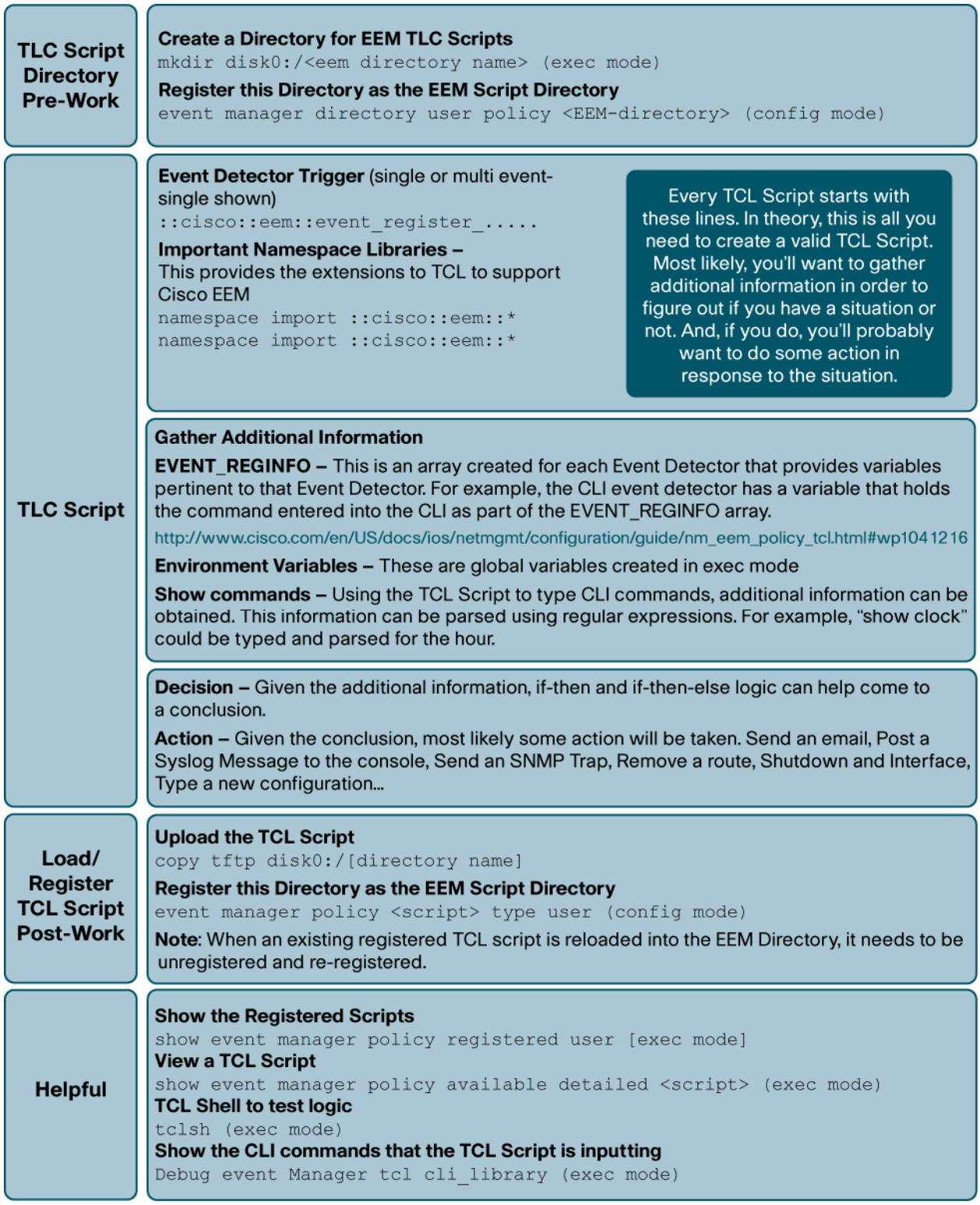
```
action_Syslog msg “OIR Event Detected.”
```

These messages can be useful when testing and debugging a script. Syslog messages can be added anywhere in a script as checkpoints to verify that a particular part if a script has been reached.

For more information about logic and for examples of actions, see the appendix.

Figure 4 summarizes TCL prework, script components, post-work, and tips.

Figure 4. TCL implementation Components and Basic Steps



Use Cases

None Event Detector

Problem Statement

When a change is made to a TCL script (a frequent occurrence when developing or testing a script), the following steps typically are performed manually:

1. Use TFTP to send the script to the device.
2. Unregister the script (if it was previously registered).
3. Reregister the script.

Suggested Solution

Use the None event detector to automate the preceding actions in a single command using the following syntax:

```
event manager run scriptload.tcl <script-to-load> [tftp-server-ip]
```

A custom command (for example, **loadscript**) can be simulated by creating an alias for the preceding command as shown in the following command:

```
alias exec loadscript event manager run scriptload.tcl
```

The syntax would now be **loadscript <script-to-load> [tftp-server-ip]**.

The environment variable **_tftp_ip** is used, and it should be set to the TFTP server's IP address. The ***tftp-server-ip*** argument is optional and can override the environment variable.

Note: The None event detector is also very useful for testing scripts since it can be triggered manually.

Script Flow

- Use **event_reqinfo** to determine the number of arguments that were passed.
  - If 2 arguments were passed, the ***script-to-load*** and ***tftp-server-ip*** values were provided.
  - If 1 argument was passed, the ***script-to-load*** value is assumed to have been provided.
  - If 0 arguments were passed, no ***script-to-load*** value was specified, and this is an error.
- Set the variable **scriptname** to **argument 1** (name of the script to load).
- Capture Cisco IOS EEM policy directory information through the CLI command **show event manager directory user policy**.
- Use TFTP to send the script and unregister and reregister the script.
- Verify script registration using the command **show event manager policy reg | i $scriptname**. This command filters output to the specific script registered.
- Output a custom syslog message with script registration information.

Test Procedure

- Add the environment variable **_tftp_ip:**
  ```
  event manager environment _tftp_ip <tftp-ip-address>
  ```
- Manually run the script using TFTP and register an updated script:
  ```
  event manager run scriptload.tcl newscript.tcl
  ```

The following syslog output shows sample results:

```
Clifton-Core6k#
1d00h: %HA_EM-6-LOG: scriptload.tcl: 27   script  user    counter             Off
Fri Apr 24 00:19:15 2009  checkcounter.tcl
```

Raw Script

```
#------------------------------------------------------------------
# EEM TCL script that automates the TFTP upload of a new script and the un-register
and register process.
# Syntax: event manager run <this-script.tcl> <script-to-load.tcl> [tftp_ip]
# tftp_ip is an optional parameter to override the environment variable
#
# Possible use with alias: alias exec loadscript event manager run this-script.tcl
#  # loadscript <script-to-load.tcl> [tftp_ip]
#
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------



# This script uses the Event Detector “None” and is run manually using the command
“event manager run”
::cisco::eem::event_register_none



### The following EEM environment variables are used:
###
### _tftp_ip             - TFTP IP address of the server with the EEM script to
upload
###                        Note: this environment variable can be overridden by an
argument after the script name



# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*



# Set array for event_reqinfo
# Array is populated with additional event information
```

```
#
http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1041216
array set arr_einfo [event_reqinfo]

# If number of arguments = 2, a TFTP server IP was specified as an argument
    if { $arr_einfo(argc) == 2 } {
   set _tftp_ip $arr_einfo(arg2)

#  Debug message to verify an TFTP IP set to argument
   action_Syslog msg “_tftp_ip set to: $arr_einfo(arg2)”
    }

# If number of arguments = 0, no script was specified
    if { $arr_einfo(argc) == 0 } {

   action_Syslog msg “ERROR: You must specify a script:  event manager run scriptload.tcl yourscript.tcl”
    }

# Check for _tftp_ip variable.  If not set using an environment variable or argument, show an error message.
    if {![info exists _tftp_ip]} {
   set result \
          “Define _tftp_ip Environment Variable first: ex. event manager envir _tftp_ip 10.10.10.1”
   error $result $errorInfo
    }

# Debug message to verify scriptname
# action_Syslog msg “Argument 1 - Scriptname is $arr_einfo(arg1)”
set scriptname $arr_einfo(arg1)



#### UPLOAD THE SCRIPT
# 1.  Open CLI, set file prompt to quiet (no prompts for TFTP)
# 2.  Set upload directory to policy file location
# 3.  Upload the script using TFTP
# 4.  Unregister script
# 5.  Register script
# 6.  Verify script registration
# 7.  Close CLI

# 1.  Open CLI
if [catch {cli_open} result] {
     error $result $errorInfo
     } else {
       array set cli1 $result
}

if [catch {cli_exec $cli1(fd) “enable”} result] {
```

```
    error $result $errorInfo
}

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

# Set file prompt quiet
if [catch {cli_exec $cli1(fd) "file prompt quiet"} result] {
    error $result $errorInfo
}
if [catch {cli_exec $cli1(fd) "exit"} result] {
    error $result $errorInfo
}

# 2.  Set variable to policy file directory.  This will be used as the TFTP
destination.
if [catch {cli_exec $cli1(fd) "show event manager directory user policy"}
eem_pol_dir] {
    error $result $errorInfo
}

# Debug messages
#action_Syslog msg "TFTP IP: $_tftp_ip"
#action_Syslog msg "scriptname: $scriptname"
#action_Syslog msg "pol dir: $eem_pol_dir"

# 3.  TFTP the script
if [catch {cli_exec $cli1(fd) "copy tftp://$_tftp_ip/$scriptname $eem_pol_dir"}
result] {
    error $result $errorInfo
}
# Debug message - show TFTP result
action_Syslog msg "TFTP result: $result"

# Enter config mode
if [catch {cli_exec $cli1(fd) "config t"} result] {
    error $result $errorInfo
}

# 4.  UNREGISTER SCRIPT
if [catch {cli_exec $cli1(fd) "no event manager policy $scriptname"} result] {
    error $result $errorInfo
}
# Debug message
# action_Syslog msg "Unregister $scriptname. $result"


# 5.  REGISTER SCRIPT
if [catch {cli_exec $cli1(fd) "event manager policy $scriptname"} result] {
    error $result $errorInfo
```

```
}
# Debug message
# action_Syslog msg “Register $scriptname. $result”


# 6.  VERIFY SCRIPT REGISTRATION
if [catch {cli_exec $cli1(fd) “exit”} result] {
    error $result $errorInfo
}
if [catch {cli_exec $cli1(fd) “show event manager policy reg | i $scriptname”}
result] {
    error $result $errorInfo
}
# Show Event Manager Policy Registration of script
action_Syslog msg “Script registered: \n $result”


# 7.  Close CLI
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}
```

Syslog Event Detector

Problem Statement

A customer wants to monitor Open Shortest Path First (OSPF) adjacency changes on a core device and to be alerted when an OSPF neighbor is added or removed.

Suggested Solution

Use the Syslog event detector to detect OSPF adjacency changes. Parse the syslog message generated by the adjacency change to determine whether a neighbor was added or removed. Then email the administrator the current OSPF neighbor information.

The Syslog event detector is easy to implement and easily understood. It can also be helpful to gather troubleshooting information about a sporadic event.

Script Flow

- Register the script to look for the pattern “%OSPF-5-ADJCHG” in the syslog.
- Use the **event_reqinfo** command to get the full syslog message.
- Capture current neighbor information using **show ip ospf neighbor**.
- Parse the syslog message to determine whether a neighbor was removed (**FULL** is changed to **DOWN**) or added (**LOADING** is changed to **FULL**).
- Save the script execution time.
- Email the administrator with the timestamp and current OSPF neighbor information.

Test Procedure

- Add the environment variable **_email_server**:

```
event manager environment _email_server <smtp.domain.com>
```

- Register the script (configuration mode):

```
event manager policy EEM-Syslog-email.tcl
```

- Add and remove an OSPF neighbor to test the script.

Here is some sample email output:

```
This email is generated by EEM.
22:23:07 UTC Fri Apr 24 2009

*Apr 24 22:23:02.052: %OSPF-5-ADJCHG: Process 1, Nbr 10.10.100.1 on
GigabitEthernet1/14 from LOADING to FULL, Loading Done

show ip ospf neighbor:


Neighbor ID     Pri   State       Dead Time   Address    Interface
10.10.100.1       1   FULL/BDR    00:00:37    10.10.10.2 GigabitEthernet1/14
```

Raw Script

```
#-------------------------------------------------------------------
# EEM policy that will monitor SYSLOG for OSPF Adjacency change.
# If change a new neighbor adjacency is formed or taken down,
# send an email alert with the time and current OSPF neighbor information.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
```

```
# All rights reserved.
#------------------------------------------------------------------

### The following EEM environment variables are used:
###
### _email_server
###                                    - A Simple Mail Transfer Protocol (SMTP)
###                                      mail server used to send e-mail.
### Example: _email_server               mailserver.example.com


# Register for a Syslog event.  Event Detector: Syslog
# Match pattern for OSPF Adjacency Change
::cisco::eem::event_register_Syslog pattern “%OSPF-5-ADJCHG”


# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*


# Set array for event_reqinfo
# Array is populated with additional event information
#
http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.h
tml#wp1041216
array set Syslog_info [event_reqinfo]
set msg $Syslog_info(msg)


# Set routername variable for use later
set routername [info hostname]


# Debug message to verify an OSPF Adjacency change was detected
# action_Syslog msg “==OSPF ADJ CHANGE DETECTED==“


# Open CLI to get show OSPF neighbors
#
# ------------------   cli open   ------------------
#
if [catch {cli_open} result] {
   error $result $errorInfo
   } else {
   array set cli $result
}
```

```
if [catch {cli_exec $cli(fd) “enable”} result] {
   error $result $errorInfo
}

# Execute CLI command and store in variable
if [catch {cli_exec $cli(fd) “show ip ospf neighbor”} ospfnei] {
   error $result $errorInfo
}

#
#--------------------- cli close ------------------------
#
cli_close $cli(fd) $cli(tty_id)


# ACTION if OSPF neighbor adjacency lost
if [regexp {FULL to DOWN} $msg] {
#  action_Syslog msg “==FULL to DOWN==“

# save exact execution time
   set time_now [clock seconds]
   set time_now [clock format $time_now -format “%T %Z %a %b %d %Y”]

#
# EMAIL MESSAGE
# This manually creates a text message with specific format to be used by the
# smtp_send_email command later to send an email alert.
#
# Ensure the following are configured:
# ip domain-name <domain.com>
#
# If a hostname is used for mailservername, ensure the following are configured:
# ip name-server <dns-server>
# ip domain-lookup
#
# NOTE: Change environment variable _email_server to your SMTP server
#
# The email below references the following variables:
#
# $routername: hostname of device
# $time_now: time when specific Syslog message was detected
# $msg: Syslog message received
# $ospfnei: output of “show ip ospf neighbor”
#
#
set email_message “Mailservername: $_email_server
From: eem@domain.com
To: group@domain.com
Cc:
Subject: EEM: OSPF - neighbor loss on $routername
```

```
This email is generated by EEM.
$time_now

$msg

show ip ospf neighbor:
$ospfnei
“

# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }

# Debug message to check email transmission status
action_Syslog msg “$result”

#Close ACTION
}



#ACTION if new OSPF neighbor adjacency detected
if [regexp {LOADING to FULL} $msg] {
   action_Syslog msg “==LOADING to FULL==“

# save exact execution time
   set time_now [clock seconds]
   set time_now [clock format $time_now -format “%T %Z %a %b %d %Y”]


#EMAIL MESSAGE
set email_message “Mailservername: $_email_server
From: eem@domain.com
To: group@domain.com
Cc:
Subject: EEM: OSPF - new neighbor adjacency on $routername

This email is generated by EEM.
$time_now

$msg

show ip ospf neighbor:
$ospfnei
“
```

```
# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }

# Debug message to check email transmission status
action_Syslog msg “$result”

#Close ACTION
}
```

SNMP Event Detector

Problem Statement

You want to monitor CPU utilization for a device (or for all devices). You want a notification to be sent if CPU utilization reaches a certain threshold, and if high CPU utilization persists, you want notifications to continue to be sent.

Suggested Solution

Use the SNMP event detector. Set it to send an email when CPU utilization reaches a certain threshold. Emails will continue to be sent as long as CPU utilization continues to stay above the threshold.

Script Flow

- Poll the SNMP MIB every 60 seconds for the CPU level.
- The SNMP event detector is triggered if CPU utilization is above 75 percent.
- Use **event_reqinfo** to get the CPU level that caused the event to trigger.
- Obtain the hostname.
- Send a syslog message to the console.
- Generate an email message to the on-call engineer.

Test Procedure

- Set the threshold to a low value such as 5 percent.
- Enter **show** or **debug** commands to trigger the SNMP MIB.

Raw Script

```
#-------------------------------------------------------------------
# Simple EEM script to Monitor a SNMP OID to determine if the CPU reached a
threshold
# EVENT DETECTOR = SNMP
#
# Trigger Tracks SNMP OID 1.3.6.1.4.1.9.2.1.58.0 for configured CPU Threshold
# When the CPU threshold is detected, send an email and post a Syslog message to
the console.
#
# Make sure the switch can route to the FQDN of the email server.
# IP HOST configuration for the email server may be required if no DNS is set up.
```

```
# SMTP requires a Domain name to be configured.
#
# TEST
# Set the threshold to something low such as 5%
# Type show commands or debug to trigger the SNMP MIB.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------


# START SCRIPT
#
http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemt.html#wp1135239


# Entry if the CPU is over 75%
# To have the trigger send an alert every poll interval, don't set an exit value.

   ::cisco::eem::event_register_snmp oid 1.3.6.1.4.1.9.2.1.58.0 get_type exact
entry_op ge entry_val 75 entry_type value poll_interval 60 maxrun 20



# ENVIRONMENT VARIABLES
# $email_server is an environmental variable -  email.abc.com
# (config)#event manager environment _email_server email.abc.com


#NAMESPACE IMPORT
#Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above. Add the following lines to your script:

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


#SET ARRAY FOR EVENT_REQINFO

   array set cpuinfo [event_reqinfo]
   set cpuvalue $infocpu(val)
   set routername [info host]


#SEND A MESSAGE TO SYSLOG ABOUT THE CPU

   action_Syslog msg “############################”
   action_Syslog msg “The CPU level is $cpuvalue”
   action_Syslog msg “############################”


# SEND AN EMAIL
```

```
# EMAIL MESSAGE
set email_message
“Mailservername: $_email_server
From: $routername@abc.com
To: on-call@abc.com
Cc:
Subject: The CPU level is $cpuvalue on $routername

This email is generated by EEM
The CPU level is $cpuvalue on $routername.
“

# SYSLOG MESSAGE NOTIFICATION ON WHETHER THE EMAIL WAS SENT SUCCESSFULLY

   if {[catch {smtp_send_email $email_message} result]} {
    set result “Email send failed”
   } else {
   set result “Email Sent”
   }

   action_Syslog msg “$result”
```

Timer Event Detector

Problem Statement

A customer needs to execute a particular CLI command at regular intervals and wants the results sent by email.

Suggested Solution

Use the Timer event detector with a cron entry to run CLI commands periodically. Email the command output to the administrator.

This example (and other examples) can be found in the Cisco IOS EEM TCL documentation: http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1039232.

This script makes extensive use of environment variables. This approach provides flexibility; the same script can be used across many devices, with customization performed at the environment-variable level.

This script also uses one of the built-in email templates (email_template_cmd.tm), which uses the input variable **cmd_output** and the environment variable **_show_cmd**. The manual email method (shown in the previous two use cases) is probably easier and provides more customization options.

Script Flow

Register the script using a cron timer

- Check whether the required environment variables exist.
- Execute the CLI command and save the timestamp.
- This script removes the trailing router prompt, which may be useful.
- Write to the log file.
- Send email to the administrator.

Test Procedure

- Define the environment variables used by the script:

```
event manager environment            _cron_entry 0-59/5 * * * *
 (this CRON entry causes the script to execute every 5 minutes)
event manager env _log_file          disk0:/eem/logfile.txt
event manager env _email_server      smtp.domain.com
event manager env _email_from        cfunakur@cisco.com
event manager env _email_to          cfunakur@cisco.com
event manager env _show_cmd          show interface f4/1

Verify values using “show event manager environment”
```

- Register the script (configuration mode):

```
event manager policy EEM-timer-script.tcl
```

- After the script is registered, periodic email messages should be received:

```
Subject: From router Core6k: Periodic show interface f4/1 Output

Timestamp: 22:08:01 UTC Wed Apr 22 2009
```

```
FastEthernet4/1 is up, line protocol is up (connected)
  Hardware is C6k 100Mb 802.3, address is 001c.0f5e.4700 (bia 001c.0f5e.4700)
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 173/255, txload 1/255, rxload 1/255
...
```

Raw Script

```
::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
###                             Min   Hr  day month day-of-week
### _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]

# Format time_now to be readable
#      (Format = 00:53:44 PDT Mon May 02 2005)
set time_now [clock format $time_now -format “%T %Z %a %b %d %Y”]

# 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
    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”
    }

set cmd_output [concat Timestamp: $time_now\n$cmd_output]

   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
    }
}
```

Interface Event Detector

Problem Statement

A customer wants to monitor a problematic interface. After output errors cross a specific threshold, an email message should be sent with **show interface** information.

Suggested Solution

Use the Interface event detector to execute when FastEthernet4/1 output_errors on reach 10. Run a **show interface** CLI command and email the command output to the administrator.

There are 22 interface counters that can be monitored (typically shown in a **show interface** command). The interface counters are described in the following documentation: http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1040117.

This script also demonstrates how another script can be launched. A secondary script is launched at the end to send the email message. The email content is stored in environment variables since local variables can be referenced only from a local script. The following environment variables are set by the primary script and referenced by the secondary script:

- **_email_subject**
- **_email_text**

The secondary script also uses environment variables to define the email server and to and from email addresses.

Script Flow

- Register the script to monitor a counter on a specific interface.
- Collect event details using **event_reqinfo**.
- Run a CLI command (use the interface name collected with **event_reqinfo**).
- Write the command output to **_email_text**.
- Write the subject line to **_email_text** (using the counter name from the **event_reqinfo** information).
- Launch the secondary script to send email.
- The secondary script sends the email message and resets the environment variables.

Test Procedure

- Define the environment variables used by the script and secondary email script:

```
event manager env _email_server      smtp.domain.com
event manager env _email_from        cfunakur@cisco.com
event manager env _email_to          cfunakur@cisco.com
event manager env _email_text        placeholder
event manager env _email_subject     placeholder
```

```
Verify values using “show event manager environment”
```

- Register the script (configuration mode):

```
event manager policy EEM-interface-callemail.tcl
```

- When error counters reach the specified value, an email message should be received.

Raw Script

```
#-------------------------------------------------------------------
# EEM policy that will monitor interface counters and generate an event when a
counter exceeds a threshold.
# If the threshold for a counter is exceeded, capture CLI output for the interface
and save to an environment variable.
# A second script is then called to send an email using the information contained
in the environment variable.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------

### The following EEM environment variables are used:
###
### _email_text (mandatory)             This script writes to an environment
variables
### _email_subject (mandatory)          Information written is then used by
another script which sends an email
###


# Register for an interface counter event.  Event Detector: interface
# Once output_errors reach 10, raise event
::cisco::eem::event_register_interface name FastEthernet4/1 parameter output_errors
entry_val 10 entry_op eq entry_type value exit_val 20 exit_op eq exit_type value



# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*



# Set array for event_reqinfo
# Array is populated with additional event information
#
http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.h
tml#wp1041216
```

```
array set event_details [event_reqinfo]


# Debug messages to verify event_reqinfo details
# Values in the array are also used later in the script
#
#action_Syslog msg “Name is $event_details(name)”
#action_Syslog msg “Parameter is $event_details(parameter)”
#action_Syslog msg “Value is $event_details(value)”



# Open CLI to get show interface information
#
# ------------------   cli open   ------------------
#
if [catch {cli_open} result] {
   error $result $errorInfo
   } else {
   array set cli $result
}

if [catch {cli_exec $cli(fd) “enable”} result] {
   error $result $errorInfo
}

# Execute CLI command and store in variable
# NOTE: the CLI command uses the interface name information from event_reqinfo
#
if [catch {cli_exec $cli(fd) “show int $event_details(name)”} result] {
   error $result $errorInfo
}

#Store command output in a local variable
set show_int $result

#Debug message to verify command output
action_Syslog msg “====show_int is $show_int”


# Change Environment Variable
if [catch {cli_exec $cli(fd) “conf t”} result] {
error $result $errorInfo
}
if [catch {cli_exec $cli(fd) “event manager environment _email_text $show_int”}
result] {
error $result $errorInfo
}
```

```
if [catch {cli_exec $cli(fd) “event manager environment _email_subject Interface
$event_details(parameter) exceeded”} result] {
error $result $errorInfo
}
#
#--------------------- cli close ------------------------
#
     cli_close $cli(fd) $cli(tty_id)



# LAUNCH SECONDARY SCRIPT TO SEND EMAIL
action_policy EEM-sendmail-subproc.tcl
```

Raw Script (Secondary Email Script)

```
#-------------------------------------------------------------------
# This EEM policy is called by another script to send an email.
# The email contains information written to the environment variable _email_text.
#
#  Requirements        :  -EEM env variables-
#                         event manager environment _email_server <your-mailserver-
ipaddress or dns-name>
#                         event manager environment _email_from <your-email-from-
address>
#                         event manager environment _email_to <your-email-to-
address>
#                         event manager environment _email_subject !This is written
by another script
#                         event manager environment _email_text !This is written by
another script
#
#                         Example: event manager environment _email_server
10.10.10.10
#                                  event manager environment _email_from router-
123@cisco.com
#                                  event manager environment _email_to
noc@cisco.com
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------

# Register a “none” event detector.  This script is called by another script.
::cisco::eem::event_register_none queue_priority low nice 1 maxrun 30
```

```
# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*


#--- Check required environment variable(s) has been defined

if {![info exists _email_server]} {
    set result “EEM Policy Error: variable $_email_server has not been set”
    error $result $errorInfo
}

if {![info exists _email_to]} {
    set result “EEM Policy Error: variable $_email_to has not been set”
    error $result $errorInfo
}

if {![info exists _email_from]} {
    set result “EEM Policy Error: variable $_email_from has not been set”
    error $result $errorInfo
}


# Set routername variable for use later
set routername [info hostname]



# EMAIL MESSAGE
# This manually creates a text message with specific format to be used by the
# smtp_send_email command later to send an email alert.
#
# Ensure the following are configured:
# ip domain-name <domain.com>
#
# If a hostname is used for mailservername, ensure the following are configured:
# ip name-server <dns-server>
# ip domain-lookup
#
#
#
set email_message “Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc:
Subject: EEM: $_email_subject from $routername
```

```
This email is generated by EEM.

$_email_text
”

# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }

# Debug message to check email transmission status
action_Syslog msg “$result”




# RESET ENVIRONMENT VARIABLE
#
#-------------------   “ cli open”   -------------------
#
if [catch {cli_open} result] {
  error $result $errorInfo
  } else {
  array set cli $result
}

if [catch {cli_exec $cli(fd) “enable”} result] {
    error $result $errorInfo
}

if [catch {cli_exec $cli(fd) “conf t”} result] {
   error $result $errorInfo
}

if [catch {cli_exec $cli(fd) “event manager environment _email_text Placeholder”}
result] {
   error $result $errorInfo
}

if [catch {cli_exec $cli(fd) “event manager environment _email_subject
Placeholder”} result] {
   error $result $errorInfo
}

#
#--------------------- cli close ------------------------
#
  cli_close $cli(fd) $cli(tty_id)
```

OIR Event Detector

Problem Statement

A customer wants to know whenever a module is added or removed from a chassis.

Suggested Solution

Use the OIR event detector to detect online insertion and removal events. If an OIR event is detected, collect information and email the current module information to the administrator.

Script Flow

- Register the script to monitor OIR events.
- Use the **event_reqinfo** command to get more information about the event (the slot and whether the event is an insertion or a removal).
- Save the timestamp for the email message.
- Determine whether the module was inserted or removed.
- Save the CLI output of **show module** or **show module *X***.
- Email the timestamp and current module inventory information to the administrator.

Test Procedure

- Register the script (configuration mode):

```
event manager policy EEM-oir.tcl
```

- Insert or remove module to test it.

Here is some sample email output:

```
Subject: EEM: Module inserted on Core6K

00:52:49 UTC Thu Apr 23 2009
Module inserted in slot 4.

Mod Ports Card Type                                Model              Serial No.
--- ----- ---------------------------------------- ------------------ -----------
  4   48  SFM-capable 48-port 10/100 Mbps RJ45   WS-X6548-RJ-45     SAD061901FX

Mod MAC addresses                       Hw    Fw           Sw           Status
--- ----------------------------------- ------ ------------ ------------ -------
  4  0001.6445.bc90 to 0001.6445.bcbf   0.501 Unknown      Unknown      PwrDeny

Mod  Online Diag Status
---- -------------------
  4  Not Applicable

Here is more sample email output:

Subject: EEM: Module removed on Core6K
```

```
00:52:09 UTC Thu Apr 23 2009
Module removed from slot 4.

Mod Ports Card Type                                Model              Serial No.
--- ----- ---------------------------------------- ------------------ -----------
  1    5  Supervisor Engine 720 10GE (Active)    VS-S720-10G        SAD112803PL
  3   16  CEF720 16 port 10GE                     WS-X6716-10GE      SAD1149028V

Mod MAC addresses                       Hw    Fw           Sw           Status
--- ----------------------------------- ------ ------------ ------------ -------
  1  000f.8f3a.9b00 to 000f.8f3a.9b07   0.702 8.5(2)       12.2(33)SXI  Ok
  3  0030.f276.5d10 to 0030.f276.5d1f   0.521 12.2(18r)S1  12.2(33)SXI  Ok

Mod  Sub-Module                  Model              Serial       Hw     Status
---- --------------------------- ------------------ ----------- ------- -------
  1  Policy Feature Card 3       VS-F6K-PFC3CXL     SAD112803N6  0.204  Ok
  1  MSFC3 Daughterboard         VS-F6K-MSFC3       SAD112804S7  0.102  Ok
  3  Distributed Forwarding Card WS-F6700-DFC3CXL   SAD114804DN  1.0    Ok

Mod  Online Diag Status
---- -------------------
  1  Pass
  3  Pass
```

Raw Script

```
#-------------------------------------------------------------------
# EEM policy that will activate on online insertion and removal (OIR) event.
# If an OIR event is detected, send an email with detail information.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#-------------------------------------------------------------------

# Register for an online insertion and removal (OIR) event.  Event Detector: oir
::cisco::eem::event_register_oir


# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
```

```
# Set array for event_reqinfo
# Array is populated with additional event information
#
http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1041216
array set arr_einfo [event_reqinfo]


# Debug information based on event_reqinfo
#action_Syslog msg “Event ID: $arr_einfo(event_id)”
#action_Syslog msg “Event Type: $arr_einfo(event_type)”
#action_Syslog msg “Event String: $arr_einfo(event_type_string)”
#action_Syslog msg “Event pub time: $arr_einfo(event_pub_time)”
#action_Syslog msg “Event pub sec: $arr_einfo(event_pub_sec)”
#action_Syslog msg “Event severity: $arr_einfo(event_severity)”
#action_Syslog msg “Event: $arr_einfo(event)”
#action_Syslog msg “Slot: $arr_einfo(slot)”


# Set variables to capture slot information, insertion type and event info
set module_slot $arr_einfo(slot)
set insertion_type $arr_einfo(event)
set event_type $arr_einfo(event_type)


# Set routername variable for use later
set routername [info hostname]


# save exact execution time
set time_now [clock seconds]
set time_now [clock format $time_now -format “%T %Z %a %b %d %Y”]


#ACTION - MODULE INSERTED
if {$insertion_type == “inserted”} {


#-----Open CLI, capture current module information
   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) “show module $module_slot”} result] {
        error $result $errorInfo
    } else {
             set cmd_output $result
             # format output: remove trailing router prompt
             regexp {\n*(.*\n)([^\n]*)$} $result dummy cmd_output
    }

#-----Close CLI
    cli_close $cli(fd) $cli(tty_id)


# EMAIL MESSAGE
# This manually creates a text message with specific format to be used by the
# smtp_send_email command later to send an email alert.
#
# Ensure the following are configured:
# ip domain-name <domain.com>
#
# If a hostname is used for mailservername, ensure the following are configured:
# ip name-server <dns-server>
# ip domain-lookup
#
# NOTE: Change environment variable _email_server to your SMTP server
#
# The email below references the following variables:
#
# $routername: hostname of device
# $time_now: time when specific Syslog message was detected
# $cmd_output: show module information
#
#
set email_message “Mailservername: smtp.domain.com
From: eem@domain.com
To: group@domain.com
Cc:
Subject: EEM: Module inserted on $routername

This email is generated by EEM.
$time_now
Module inserted in slot $module_slot

$cmd_output
“

# Send email message
    if {[catch {smtp_send_email $email_message} result]} {
       set result “Email send failed”
       } else {
       set result “Email Sent”
```

```
    }

# Debug message to check email transmission status
action_Syslog msg “$result”

#Close ACTION
}



#ACTION - MODULE REMOVED
if {$insertion_type == “removed”} {


#-----Open CLI, capture current module information
    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) “show module”} result] {
        error $result $errorInfo
    } else {
             set cmd_output $result
             # format output: remove trailing router prompt
             regexp {\n*(.*\n)([^\n]*)$} $result dummy cmd_output
    }

#-----Close CLI
    cli_close $cli(fd) $cli(tty_id)

# EMAIL MESSAGE
set email_message “Mailservername: smtp.domain.com
From: eem@domain.com
To: group@domain.com
Cc:
Subject: EEM: Module removed on $routername

This email is generated by EEM.
$time_now
Module removed from slot $module_slot

$cmd_output
“
```

```
# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }

# Debug message to check email transmission status
action_Syslog msg “$result”

#Close ACTION
}
```

GOLD Event Detector

Problem Statement

When a Cisco GOLD test runs, you want a notification sent if an error with a major or minor severity level needs attention.

Suggested Solution

Using the GOLD event detector, determine whether a major or minor error has occurred and send an email message to the operations alias.

Script Flow

- Use the GOLD event detector to trigger when an error with a major or minor severity level occurs (use the normal severity level for testing purposes).
- Use **event_reqinfo** to get the event severity level, the card, and the test type.
- Obtain the hostname.
- Send a syslog message to the console.
- Generate an email message to the operations team.

Test Procedure

- Add **normal** to the event detector.
- The addition of **normal** enables the event detector to trigger.
- Run **diagnostic start module [x] tst testeobcstressping**.
- This test will return a severity level of normal.

Raw Script

```
#------------------------------------------------------------------
# Simple EEM script to Monitor GOLD Tests
# EVENT DETECTOR = GOLD
#
# Trigger Tracks GOLD Tests and looks for Major and Minor issues.
#
# Logic
# Purpose of Script-Use the GOLD Detector to monitor .
# If it does, send an email alert as well as a Syslog to the console.
#
# TEST
# diagnostic start module 2 test testeobcstressping
# The sweep-ping process pings the module with 20,000 SCP-ping packets
# With the Test, severity_normal was put in.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------


#START SCRIPT
#http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemt.html#wp1179103


   ::cisco::eem::event_register_gold card all severity_major severity_minor maxrun
20



# ENVIRONMENT VARIABLES
# set environmental variable for _email_server for the email
```

```
# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above. Add the following lines to your script:

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

# SET ARRAY TO GET EVENT_REQINFO TO FIND PROCESS NAME

   array set info [event_reqinfo]
   set card $info(card)
   set severity $info(event_severity)
   set testtype $info(tt)
   set routername [info host]

#SYSLOG MESSAGE

   action_Syslog msg
"######################################################################"
   action_Syslog msg
"######################################################################"
   action_Syslog msg "GOLD"
   action_Syslog msg "GOLD Testing on $routername revealed an $severity error
severity,
   action_Syslog msg "Testing Type $testtype on Card $card"
   action_Syslog msg
"######################################################################"
   action_Syslog msg
"######################################################################"

#SEND AN EMAIL

   #EMAIL MESSAGE
set email_message "Mailservername: $_email_server
From: $routername@abc.com
To: tech_ops@abc.com
Cc:
Subject: GOLD Test Error on $routername

This email is generated by EEM - GOLD Testing on $routername revealed an $severity
error severity,
Testing Type $testtype on Card $card".
"

   if {[catch {smtp_send_email $email_message} result]} {
    set result "Email send failed"
   } else {
    set result "Email Sent"
   }

   action_Syslog msg "$result"
```

Process Event Detector

Problem Statement

With Cisco IOS Software Modularity, processes can be restarted. When a process restarts, notification other than syslog needs to be sent to the on-call engineer.

Suggested Solution

Using the Process event detector, send an email message to the on-call engineer to let the engineer know the process that restarted.

Script Flow

- Use the Process event detector to trigger when any process in Cisco IOS Software Modularity restarts.
- Use **event_reqinfo** to get the process that restarted.
- Obtain the hostname.
- Send a syslog message to the console.
- Generate an email message to the on-call engineer.

Test Procedure

- In execution mode, enter a **show process cpu** command to obtain a list of processes.
- In execution mode, enter the command **process restart [process name]**.

Raw Script

```
#-------------------------------------------------------------------
# Simple EEM script to Monitor a Modular Process Restart
# EVENT DETECTOR = PROCESS/SYSTEM MANAGER
#
# Trigger Tracks Modular Processes on the Switch.
# A process is restarted.
# The Detector is triggered.
# The process the name is taken from event_reqinfo and placed in the Syslog message
and an email is sent
# with the same information.
#
# Make sure the switch can route to the FQDN of the email server.
# IP HOST configuration for the email server may be required if no DNS is set up.
# SMTP requires a Domain name to be configured.
#
# TEST
# c6K#process restart <process name>
# c6K#show proc cpu (to view processes)
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
```

```
#-------------------------------------------------------------------

# START SCRIPT
#
http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemt.html#wp1178468

# PROCESS/SYSTEM MANAGER EVENT DETECTOR

   ::cisco::eem::event_register_process user_restart maxrun 20


# ENVIRONMENT VARIABLES
# $email_server is an environmental variable -  email.abc.com
# (config)#event manager environment _email_server email.abc.com

# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code such as the first line of the
# code above. Add the following lines to your script:

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

#SET ARRAY TO GET EVENT_REQINFO TO FIND PROCESS NAME AND THE ROUTER NAME

   array set info [event_reqinfo]
   set process $info(process_name)
   set id $info(instance)
   set routername [info host]


# SYSLOG MESSAGE

   action_Syslog msg "#################################################################"
   action_Syslog msg "The Process ID $id ($process) has restarted on $routername"
   action_Syslog msg "#################################################################"

# SEND AN EMAIL

# EMAIL MESSAGE
set email_message
"Mailservername: $_email_server
From: $routername@abc.com
To: emar@cisco.com
Cc:
Subject: The Process ID $id ($process) has restarted on $routername

This email is generated by EEM
Process $process Restarted on $routername.
"
```

```
# SYSLOG MESSAGE NOTIFICATION ON WHETHER THE EMAIL WAS SENT SUCCESSFULLY

   if {[catch {smtp_send_email $email_message} result]} {
    set result “Email send failed”
   } else {
   set result “Email Sent”
   }

   action_Syslog msg “$result”
```

WDSYSMON Event Detector

Problem Statement

A customer is concerned about occasional CPU utilization spikes and wants to know what processes may be running during the high CPU utilization condition.

Suggested Solution

Use the WDSYSMON event detector to monitor total CPU utilization and execute a script if CPU utilization exceeds 70 percent. If the script triggered, collect CPU process information and email it to the administrator.

WDSYSMON event detector registration also allows logical operations on a combination of subevents within a given time window. Subevents can include CPU and memory utilization for specific processes. For more information, see the documentation at http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1040997.

Script Flow

- Register the script to monitor total CPU utilization over 70 percent.
- Save the timestamp for the email message.
- Save the CLI output of **show process cpu | exclude 0.0**.
- Email the timestamp and current CPU process information to the administrator.

Test Procedure

- Register the script (configuration mode):

```
event manager policy EEM-wdsysmon-cpu.tcl
```

- Increase the CPU utilization (for testing, the threshold was set to 5 percent, and simple **show** commands were run to increase the CPU utilization).

Here is some sample email output:

```
Subject: High CPU condition on Core6K

This email is generated by EEM on high CPU condition.

21:54:13 UTC Thu Apr 23 2009

CPU utilization for five seconds: 18%/0%; one minute: 12%; five minutes: 6%
PID       5Sec    1Min     5Min Process
1         5.8%    0.8%     0.4% kernel
16407     2.6%    8.4%     4.2% ios-base
16431     0.3%    1.5%     0.6% iprouting.iosproc
16432     0.2%    0.2%     0.2% cdp2.iosproc
```

Raw Script

```
#------------------------------------------------------------------
# EEM policy that will monitor watchdog system for a system event.
# If the cpu_tot (total system CPU usage) subevent exceeds 70%,
# send an email alert with the time and current CPU process information.
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------

# Register for a Watchdog system monitor event.  Event Detector: wdsysmon
# Match on subevent cpu_tot.  If over 70%, raise event.
::cisco::eem::event_register_wdsysmon timewin 60 sub1 cpu_tot op ge val 70



# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code
such as the first line of the code above.
# Add the following lines to your script:
```

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

# save exact execution time
set time_now [clock seconds]
set time_now [clock format $time_now -format “%T %Z %a %b %d %Y”]

# Set array for event_reqinfo
# Array is populated with additional event information
#
http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.h
tml#wp1041216
array set event_details [event_reqinfo]

# Debug messages to verify the event was detected
action_Syslog msg “sub1 is $event_details(sub1)”
action_Syslog msg “CPU over 70%”

# Set routername variable for use later
set routername [info hostname]

# Open CLI to show CPU process usage
#
# -------------------   “ cli open”   -------------------
#
if [catch {cli_open} result] {
   error $result $errorInfo
   } else {
   array set cli $result
}

if [catch {cli_exec $cli(fd) “enable”} result] {
   error $result $errorInfo
}

# Execute CLI command and store in variable
if [catch {cli_exec $cli(fd) “show proc cpu | e 0.0”} cpustats] {
   error $result $errorInfo
}

#
#---------------------- cli close -----------------------
#
cli_close $cli(fd) $cli(tty_id)



#
# EMAIL MESSAGE
# This manually creates a text message with specific format to be used by the
```

```
# smtp_send_email command later to send an email alert.
#
# Ensure the following are configured:
# ip domain-name <domain.com>
#
# If a hostname is used for mailservername, ensure the following are configured:
# ip name-server <dns-server>
# ip domain-lookup
#
# NOTE: Change environment variable _email_server to your SMTP server
#
# The email below references the following variables:
#
# $routername: hostname of device
# $time_now: time when specific Syslog message was detected
# $event_details(sub1): detail information about the event
# $cpustats: output of “show proc cpu | e 0.0”
#
#
set email_message “Mailservername: $_email_server
From: eem@domain.com
To: group@domain.com
Cc:
Subject: High CPU condition on $routername

This email is generated by EEM on high CPU condition.

$time_now

$cpustats
”


# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }

# Debug message to check email transmission status
action_Syslog msg “$result”
```

Track Event Detector

Problem Statement

When a server resource fails, the static IP route advertising its IP address needs to be removed.

Suggested Solution

Use the Track event detector to track a Cisco IOS IP SLA probe to the server. If the probe fails, remove the static IP route from the routing table. This process is similar to a route health injection.

Script Flow

- Use the Cisco IOS IP SLA to probe a server. In this example, ping is used. The probe could be any probe such as TCP connect.
- Use the Track event detector to track the Cisco IOS IP SLA probe.
- If the server fails, a syslog message is sent to the console, and the static IP route is removed.
- When the server comes back up, a syslog message is sent to the console to report that the server is back up. Code can be written to reinstate the static IP route automatically, but in this case a message reporting that it came back up is all that is provided. Most likely, you will want to reinstate the IP route manually after you are sure that the server is healthy.

Test Procedure

- Set the threshold to a low value such as 5 percent.
- Enter **show** or **debug** commands to trigger the SNMP MIB.

Raw Script

```
#-------------------------------------------------------------------
# Simple EEM script to Monitor a Server Resource and De-install a Static Route
# EVENT DETECTOR = TRACK
#
# Trigger Tracks an IOS IP SLA Probe.
# Logic
# When the ping fails to the Server Resource, IOS IP SLA will set track to down.
# The probe could be more sophisticated using something like a TCP probe.  Ping was
used just for example.
# The EEM Track Event Detector tracks the IOS IP SLA tracker for up/down.
# When the Server goes from up to down, a Syslog is sent and the host ip route to
the server is removed.
# When the Server goes from down to up, a Syslog is sent that the Server is back
and to reinstate the IP Route.
# Reinstatement of the host ip route could have been part of the TCL Script, but it
makes more sense
# to put the host ip route when the operations decides the Server is ready to
alleviate any flapping.
#
# IP SLA CONFIGURATION
# IP SLA is set up to probe the server (ping, tcp connect, udp...)
# IP SLA monitor schedule is set up to track “up or down” of the probe.
# EEM Track can only track up and down (EEM IP SLA has more granularity, but is not
yet available on 12.2SX)
# IP SLA Schedule starts the monitor to track the probe.
# EEM Track then tracks the IP SLA Monitor for an UP/DOWN status.

# IP SLA Configuration
# http://www.cisco.com/en/US/docs/ios/12_4/ip_sla/configuration/guide/hstcpc.html
# enable
# config t
# ip sla monitor 42
# icmp-echo <ip address of target?

# ip sla monitor schedule
```

```
# track 42 rtr 42 state - tracks as up or down
# ip sla schedule 42 start-time now life forever

# IP SLA TROUBLESHOOTING
# show ip sla statistics 42
# show track 42
#
# TEST - no ip sla schedule 42 start-time now life forever
#
# April 2009
# Clifton Funakura (cfunakur@cisco.com)
# Erick Mar (emar@cisco.com)
#
# Copyright (c) 2009 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------



#START SCRIPT
#http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl_ps6441_TSD_Products_Configuration_Guide_Chapter.html#wp1040958
#STATE DOWN MEANS IT MUST GO FROM UP TO DOWN. JUST BEING DOWN WILL NOT TRIGGER.

# TRACK EVENT DETECTOR

   ::cisco::eem::event_register_track 42 state any maxrun 20

#ENVIRONMENT VARIABLES
#None - this script does not use any environment variables

# NAMESPACE IMPORT
# Imports Cisco specific TCL extensions that can be used in the body of the code such as the first line of the code above.
# Add the following lines to your script:

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

#SET ARRAY OF CAPTURED EVENT_REQINFO FROM THE EVENT DETECTOR

   array set trackinfo [event_reqinfo]
   set state $trackinfo(track_state)
   action_Syslog msg “Tracking state has changed to $state”

#http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemt.html#wp1050955
#track_state - State of the tracked object when the event was triggered; valid states are up or down.
```

```
#DECISION ON WHETHER THE SERVER HAS GONE FROM UP TO DOWN, OBVIOUSLY IT HAS
OTHERWISE THE SCRIPT WOULD NOT HAVE TRIGGERED.
#IF THE STATE GOES FROM UP TO DOWN, SEND SYSLOG WARNING, THEN REMOVE THE IP HOST
ROUTE

   if { $state == “down”} {

#SEND SYSLOG WARNING

   action_Syslog msg “Server1 is $state.  Removing the Host Route....”

#REMOVE IP HOST ROUTE
   #OPEN CLI
   if [catch {cli_open} result] {
        error $result $errorInfo
     } else {
         array set cli1 $result
       }

   #ENABLE
   if [catch {cli_exec $cli1(fd) “enable”} result] {
      error $result $errorInfo
   }

   #CONFIG T
   if [catch {cli_exec $cli1(fd) “config t”} result] {
       error $result $errorInfo
   }

   #REMOVE IP ROUTE TO SERVER - THE IP ADDRESS ARE REMOVED
   if [catch {cli_exec $cli1(fd) “no ip route 192.168.1.172 255.255.255.255
ethernet0”} result] {
       error $result $errorInfo
   }

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

   action_Syslog msg “The Host Route has been Removed....”
   action_Syslog msg “When Server1 is restored, please reinstate the host route
manually....”



   } else {

#ELSE, THE EVENT DETECTOR WAS TRIGGERED BY A DOWN TO UP CHANGE.
#IN THIS CASE, SEND A MESSAGE THAT THE SERVER IS BACK UP.
```

```
   action_Syslog msg “Server1 is back $state, please reinstate the host route
manually when the time is appropriate....”


   }
```

Conclusion

[[I RECOMMEND ADDING A CONCLUDING PARAGRAPH]]

For More Information

- Cisco IOS Embedded Event Manager (launch page with links to documentation, data sheets, case studies, and Q&A): http://www.cisco.com/go/eem
- Cisco IOS EEM scripting community: http://www.cisco.com/go/ciscobeyond
- Cisco IOS EEM overview (updated information regarding Cisco IOS EEM, platform support, and available event detectors): http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemo.html
- Writing Cisco IOS EEM applets using Cisco IOS Software: http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemc.html
- Writing Cisco IOS EEM policies using TCL (documentation page): http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html
- Cisco IOS Software diagnostic tools for commercial use: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps9421/prod_qas0900aecd807213d0.html

Appendix

Detecting Multiple Events

Cisco IOS EEM Version 2.4 adds multiple-event detection, which provides more control over when an applet or script runs. Up to six event statements can be used in an applet, and up to eight event statements can be used in a TCL script. The following example from the documentation shows how an applet can be configured to run only if all three events occur within 1 hour:

```
event manager applet example
event tag e1 Syslog pattern “Syslog_message”
event tag e2 snmp oid 1.2.3.4 get-type exact entry-op gt entry-val 1000 poll-interval 10
event tag e3 ioswdsysmon sub1 cpu-proc taskname “BGP” op gt val 30 triggerperiod 600
trigger occurs 1 period 3600
correlate event e1 and event e2 and event e3
action 01.0 Syslog msg “All three events occurred in an hour.”
```

Similar tag, trigger, and correlate commands are available for TCL scripts:

```
::cisco::eem::event_register_xxx tag 1 ...
::cisco::eem::event_register_yyy 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
} [occurs <occurs-val>] [period <period-val>] \
[period-start <period-start-val>] [delay <delay-val>]
```

For more information, see Cisco IOS Embedded Event Manager Version 2.4 Expanded Capabilities at http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6815/whitepaper_c11-492226.html.

Using event_reqinfo

When an event is detected and a policy is triggered, you can use the powerful variable **event_reqinfo** to obtain details about the event. Simply add the following line in your TCL script:

```
array set arr_einfo [event_reqinfo]
```

You can now reference the array of information that is provided about the event. Each event detector populates the array (in the example here, the array name is arr_einfo) with event-specific data. For example, the OIR event detector populates the array with the information shown in Table 1.

Table 1. OIR Event Detector Detail Information

Event Type	Description
event_id	Unique number that indicates the ID for this published event; multiple policies may be run for the same event, and each policy will have the same event ID
event_type	Type of event
event_type_string	ASCII string that represents the name of the event for this event type
event_pub_sec	
event_pub_msec	Time, in seconds and milliseconds, when the event was published to Cisco IOS EEM
slot	Slot number for the affected card
event	A string, removed or online, that represents either an OIR removal event or an OIR insertion event

The values in the array can be referenced by name. For example, to show which slot caused an OIR event, you can output a syslog message:

```
action_Syslog msg “Slot: $arr_einfo(slot) caused an OIR event”
```

To see detail information about the event (whether a module was removed or inserted), you can enter:

```
action_Syslog msg “Module event info: $arr_einfo(event)”
```

For information about **event_reqinfo** detail information for each event detector, see http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_eem_policy_tcl.html#wp1041216.

You can also use online help to see what array names are available using **event_reqinfo**. Enter the following command at the CLI prompt (privileged execution mode):

```
show event manager detector <event-detector> detailed
```

Here is some sample output for OIR:

```
6k2#show event manager detector oir detailed
No.  Name                Version   Node        Type
1    oir                 01.00     node5/1     RP

        TCL Configuration Syntax:
        ::cisco::eem::event_register_oir
                 [tag <tag-val>]
                 [queue_priority {normal | low | high | last}]
                 [maxrun <sec.msec>] [nice {0 | 1}]

        TCL event_reqinfo Array Names:
        event_id
        event_type
        event_type_string
        event_pub_time
        event_pub_sec
        event_pub_msec
        event_severity
```

```
event
slot

Applet Configuration Syntax:
[ no ] event [tag <tag-val>] oir
         [maxrun <sec.msec>]

Applet Built-in Environment Variables:
$_event_id
$_event_type
$_event_type_string
$_event_pub_time
$_event_pub_sec
$_event_pub_msec
$_event_severity
$_oir_event
$_oir_slot
```

Applying Decision Logic

After information parsing, often some logic needs to be applied to the information to determine what action to take.

You can use a conditional loop to determine whether the situation warrants action. (Many other types of decision logic are also available, such as While and For logic; If-Then-Else is used here as an example.)

If-Then Example

If-Then logic provides a means of determining whether a situation warrants action. Essentially, the **if** statement is tested for validity. An **if** statement that is valid is considered to be true. An **if** statement that is true leads to the action in the **then** statement. If an **if** statement is false, no action is taken.

For example, If-Then logic could be applied to a Syslog event detector looking for downed interfaces. After a syslog message containing **LINK-3-UPDOWN** is detected, you can use regular expressions to determine which interface went down.

Here is the logic:

IF the interface is the uplink, THEN shut the port connected to Server1 to make Server1 migrate to its backup interface.

Without any more logic, a false **if** statement means “do nothing.”

Note: This example does not show how the interface was parsed.

```
if { $result == “interface gigabitethernet 4/1”} {
   #OPEN CLI
   if [catch {cli_open} result] {
        error $result $errorInfo
     } else {
         array set cli1 $result
       }

   #enable
```

```
    if [catch {cli_exec $cli1(fd) “enable”} result] {
        error $result $errorInfo
    }

    #config t
    if [catch {cli_exec $cli1(fd) “config t”} result] {
        error $result $errorInfo
    }

    #interface fa3/4
    if [catch {cli_exec $cli1(fd) “interface fa3/4”} result] {
        error $result $errorInfo
    }

    #shut interface
    if [catch {cli_exec $cli1(fd) “shut”} result] {
        error $result $errorInfo
    }

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

You would probably also want to send an email message or post a syslog message, but that is beyond the scope for this example.

If-Then-Else Example

If-Then-Else logic can be written for a GOLD event detector to determine whether a major, minor, or normal event has occurred.

Here is the logic:

IF a major event occurred, THEN send an email message to the whole operations team. ELSE, IF a minor event occurred, THEN send an email message only to the on-duty engineer mailer. ELSE, IF a normal event occurred, THEN post a syslog message.

This example does not show the source of the severity level, email server environment variable, hostname variable, or source of the diagnostic test. The GOLD event detector TCL script example in the appendix shows how all these are obtained. [[SOMETHING IS WRONG HERE; THIS IS THE APPENDIX. WHAT EXAMPLE DO YOU MEAN?]]

```
if { $severity == “major”} {
    #EMAIL MESSAGE
set email_message “Mailservername: $_email_server
From: SFA1003@abc.com
To: IT_OPS@abc.com
Cc:
Subject: $hostname:  Major GOLD Event Diagnostic $diag.
This email is generated by EEM –
“
```

```
   } else {
         if { $severity == “minor”} {
          #EMAIL MESSAGE
set email_message “Mailservername: $_email_server
From: SFA1003@abc.com
To: On-Call@abc.com
Cc:
Subject: $hostname:  Minor GOLD Event Diagnostic $diag.
This email is generated by EEM –
“
   } else {
         action_Syslog msg “GOLD Diagnostic $diag was run with Normal Results”
}
}
```

Applying Actions

The previous section showed examples of some actions such as creating syslog messages and sending email messages. This section looks at these actions again as well as the action of sending an SNMP trap from within a script.

Send a Syslog Message

Syslog is a common means of sending a message alert to the console. Since the syslog message can be customized by a script, the output of a syslog message can read any way that you want.

Here is a simple example that uses the Syslog event detector to look for a syslog message:

```
::cisco::eem::event_register_Syslog pattern %LINK-3-UPDOWN: Interface Vlan66,
change state to down” maxrun 60
```

When this syslog message is sent to the console, the event detector will be triggered:

```
1d22h: %LINK-3-UPDOWN: Interface Vlan66, changed state to down
```

The action could provide more information from syslog:

```
action_Syslog msg “VLAN 66 is the primary SAP Server Farm.  The Secondary Farm
should now be active”
```

Send an Email Alert

One of the most useful functions that Cisco IOS EEM provides is the capability to send email alerts in response to detected events. The **smtp_send_email** function requires text input in a specific format. Following is an example of an email message created manually within a TCL script:

```
#EMAIL MESSAGE
#Indentation can not be used for better readability.
set email_message “Mailservername: smtp.abc.com
From: SFA1003@abc.com
To: IT_OPS@abc.com
Cc:
Subject: VLAN66-Primary SAP Subnet is DOWN.
```

```
This email is generated by EEM –
“
```

The email message can now be sent using the following script:

```
# Send email message
   if {[catch {smtp_send_email $email_message} result]} {
      set result “Email send failed”
      } else {
      set result “Email Sent”
   }
```

When sending email messages from a script, note the following considerations:

- Make sure the following setting is configured on the device:

  ```
  ip domain-name <domain.com>
  ```

- If a hostname is used for mailservername, make sure that the following settings are configured:

  ```
  ip name-server <dns-server>
  ip domain-lookup
  ```

- Do not indent the email message text using tabs for better readability. The tab characters may interfere with the message format.

Send an SNMP Trap

Cisco IOS EEM supports the sending of custom SNMP traps. Using this action, Cisco IOS EEM can send a custom message to any SNMP agent. The SNMP trap that is generated uses the Cisco IOS EEM MIB: CISCO-EMBEDDED-EVENT-MGR-MIB.my. See http://www.cisco.com/en/US/docs/ios/ha/command/reference/ha_a1.html#wp1033891.

The following example uses TCL to send a custom SNMP trap text string:

```
action_snmp_trap strdata “ VLAN66-Primary SAP Subnet is DOWN.”
```

Applets use similar syntax:

action *label* **snmp-trap** [**intdata1** *integer*] [**intdata2** *integer*] [**strdata** *string*]

For more information about the MIB, use the Cisco MIB locator at http://www.cisco.com/go/mibs.

1. For information about the structure of the MIB, go to ftp://ftp.cisco.com/pub/mibs/v2/CISCO-EMBEDDED-EVENT-MGR-MIB.my.

TCL Code Snippets

When you are creating a new TCL script, you can copy and paste commonly used script components. Table 2 provides some examples of commonly used functions.

Table 2. Common TCL Script Functions

Function	Code

Function	Code
Import namespace	`namespace import ::cisco::eem::*` `namespace import ::cisco::lib::*`
Set hostname	`set routername [info hostname]`

Open CLI	`if [catch {cli_open} result] {`
	`error $result $errorInfo`
	`} else {`
	`array set cli $result`
	`}`

Invoke **enable** as a CLI command	`if [catch {cli_exec $cli(fd) "enable"} result] {`
	`error $result $errorInfo`
	`}`

Use **show** commands	`if [catch {cli_exec $cli(fd) "enable"} result] {`
	`error $result $errorInfo`
	`}`
	`if [catch {cli_exec $cli(fd) "show version"} result] {`
	`error $result $errorInfo`
	`}`
	`set show_version $result`

Configure term [[PLS CLARIFY WHAT YOU MEAN HERE]]	`if [catch {cli_exec $cli(fd) "conf t"} result] {`
	`error $result $errorInfo`
	`}`

Close CLI	`cli_close $cli(fd) $cli(tty_id)`
Output syslog	`action_Syslog msg "variable is $localtext"`

Get **event_reqinfo** information	`array set event_details [event_reqinfo]`
	
	Set variables using array information:
	`set var $event_details(array_label)`
	
	See the following URL for array labels: http://www.cisco.com/en/US/docs/ios/12_2sx/sw_modularity/configuration/guide/nmb_eemt.html#wp1050955

Send email (manual)	`set email_message "Mailservername: $server`
	`From: $from`
	`To: $to`
	`Cc: $cc`
	`Subject: $subject`
	`$body"`
	`if {[catch {smtp_send_email $email_message}`
	`result]} {`
	`set result "Email send failed"`
	`} else {`
	`set result "Email Sent"`
	`}`

Applying Regular Expressions

Regular expressions (often abbreviated as regex) provide a means for parsing information from a text string. A regular expression matches a pattern of characters in a given string (a set of characters). Regular expressions offer flexibility, with a single regular expression matching multiple patterns.

A regular expression can be simple, matching an exact pattern. For example, a regular expression comparing the character string “dog” will match “dog” in the target string “It is raining cats and ***dogs***.” Note that regular expressions are case sensitive, so “dog” will match “dog,” but not “Dog.” This is a literal pattern match. It matches a “d” followed by an “o” followed by a “g.”

More often, a regular expression will use special characters, called metacharacters, to match a pattern. Examples [[PLEASE BE SURE THAT THESE ARE INDEED JUST EXAMPLES AND NOT THE ENTIRE LIST OF METACHARACTERS]] of metacharacters are the asterisk (*), caret (^), dollar sign ($), and dot (.). These metacharacters have special meanings. For an example, a dot means match on any single character. A caret means start the comparison before the first character in the string. A dollar sign performs a match on the last character in the string. The regular expression “^a” matches “a” in the string “abc.” Similarly, the expression “$c” matches “c” in the string “abc.”

A full description of regular expressions is beyond the scope of this document, but a number of excellent tutorials on regular expressions are available on the web, and these are searchable using any of the common web search engines.

Regular expressions are very useful in Cisco IOS EEM TCL scripts. Variables provided in the event_reqinfo array often need to be parsed to obtain enough information to make an informed decision about what action needs to be taken.

For example, the Syslog event detector is triggered by the string “%sysdown” in a system-generated syslog message. To decide what action to take, the entire syslog message needs to be parsed to determine which interface went down.

A regular expression can be applied to this entire syslog-generated message to find the interface name.

The following is an example of the entire syslog message:

```
1d20h: %LINK-3-UPDOWN: Interface Ethernet2/3, changed state to up
```

The regular expression is run against the syslog message, extracting the interface from the syslog as follows:

```
regexp {([^ ]*.Ethernet.*[0-9])} $msg intf
```

In this example, the entire syslog message is held in the variable **$msg**. The result of the regular expression run against the entire syslog message **($msg**) is “Interface Ethernet2/3.” The variable **intf** is set to this value, which can then be used later in the script to help determine appropriate actions to take.

Annotated TCL Script Example

Table 3 presents an example of a TCL script. Along the right side is annotation explaining the lines of the script.

Table 3. Annotated TCL Script

```
#Logic
#The # at the beginning of the line denotes a remark or comment that is not part of
the commands in the TCL script.
```

#User types a command with “debug” in it. #If the command is “show debugging” or “undebug all,” allow the command to execute, #Else use CLI to perform a “show clock,” to determine what time it is, #If the time is between 9am-5pm local time, disallow the command and send a Syslog alert.	
::cisco::eem::event_register_cli sync yes pattern “debug” maxrun 20	This line creates an event detector for the CLI, where it is triggered when a user types any command with **debug** in it. Setting **sync** to **yes** enables the command to be executed, but the logic will really decide whether the command is ultimately permitted.
namespace import ::cisco::eem::* namespace import ::cisco::lib::*	This code is required for every Cisco IOS EEM TCL script. It imports libraries specific to Cisco IOS EEM extending the capabilities of standard TCL.
array set cli [event_reqinfo]	This line creates the array called **cli** from the **event_reqinfo** information.
set command $cli(msg)	This line sets a variable called **command** to the **msg** variable in the array **cli**.

if { $command == “show debugging” || $command == “undebug all”} {
exit 1
} else {
	This line is If-Then logic that determines whether the **$command** variable is either **show debugging** or **undebug all.** If so, let the command execute. If not, do nothing and move on to the next line in the script.
	Note: The system sees the entire command. So if an user types **sh debug**, the system will actually see **show debugging**. You can complete the commands by pressing Tab while typing the shortcuts to see the entire command.

#OPEN CLI
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}
	This line opens the CLI in execution mode.

if [catch {cli_exec $cli1(fd) “enable”} result] {
error $result $errorInfo
}
	This line types the command **enable**.

#GET SHOW CLOCK
if [catch {cli_exec $cli1(fd) “show clock”} result] {
error $result $errorInfo
}
	This line types the command **show clock**. The variable **result** is set to the output of **show clock**.
	The **show clock** command sends this output to the console:
	*12:20:19.187 PST Fri May 1 2009

#MATCH ON THE HOUR regexp { [0-9][0-9]} $result hour	The hour is needed to determine whether the time is between 9 a.m. and 5 p.m. local time. This line runs a regular expression against the **show clock** output (the entire output is **$result**). The result of the regular expression run against **$result** is used to set the variable **hour**.

action_Syslog msg “The hour is $hour”	This line is an action to type the syslog message “The hour is $hour.” Remember that **$hour** was the variable set from the regular expression run against **$result**. The variable **result** often is used as a variable within the system. Consequently, set a **result** variable to another variable name (in this case, **hour**) before another line in the TCL scripts overwrites the variable **result**.

#DETERMINE IF THE TIME IS WITHIN RANGE
if { $hour >=9 && $hour<=24} {
action_Syslog msg “Debug is not permitted between 9am-5pm Local Time”
 	
exit 0
 	
} else {
 	
exit 1 }
 	
#CLOSE CLI
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
error $result $errorInfo
}
}
	This line uses If-Then-Else logic to decide whether the hour is within the time frame in which **debug** commands are denied.
	
	IF it is, THEN send a syslog message to the console notifying the user that the command is denied. Also, set **exit 0**, which denies the command.
	
	ELSE set **exit 1**, which lets the command run as originally typed.
	
	This line closes the CLI. Always close the CLI.

Testing the CLI TCL Script	
c6k1# c6k1#show clock *11:20:19.187 PST Fri May 1 2009 c6k1# c6k1# c6k1# c6k1#debug ip address c6k1# c6k1# 1d21h: %HA_EM-6-LOG: cli2.tcl: The hour is 11 1d21h: %HA_EM-6-LOG: cli2.tcl: Debug is not permitted between 9am-5pm Local Time c6k1# c6k1# c6k1# c6k1#undebug all All possible debugging has been turned off c6k1# c6k1# c6k1# c6k1#	From the CLI, type **show clock** to determine the time. The time is within the restricted time. From the CLI, type **debug ip address**. Because this entry is made during the restricted time, the resulting syslog message is sent to the console, and the command is not permitted. From the CLI, type **undebug all**. The command is allowed, and the system syslog message notifies the console that all debugging has been turned off.

Applet and TCL Script Comparison Example

Table 4 presents sample code for a Process event detector using both an applet and TCL script.

Table 4. Cisco IOS EEM Process Event Detector Using Both an Applet and a TCL Script

# ENVIRONMENT VARIABLES # $email_server is an environmental variable - email.abc.com # (config)#event manager environment _email_server email.abc.com	# ENVIRONMENT VARIABLES # $email_server is an environmental variable - email.abc.com # (config)#event manager environment _email_server email.abc.com	Both samples use the email.abc.com fully qualified domain name (FQDN) instead of the IP address of the mail server. Make sure that Domain Name System (DNS) is configured to resolve the FQDN, or use **ip host** to configure it manually on the switch.
event manager applet process_sysmgr		This line in the applet creates the applet; the TCL script has no corresponding line.

	# NAMESPACE IMPORT # Imports Cisco specific TCL extensions that can be used in the body of the code such as the first line of the # code above. Add the following lines to your script: namespace import ::cisco::eem::* namespace import ::cisco::lib::*	The TCL script requires the importation of the Cisco libraries, Applets are embedded in Cisco IOS Software, so there is no need to import any additional libraries.
event process user-restart path . maxrun 20	# PROCESS/SYSTEM MANAGER EVENT DETECTOR ::cisco::eem::event_register_process user_restart maxrun 20	These equivalent lines create the event detector triggers. For the applet, the path is a regular expression, so "." means any process.
action 1.0 info type routername	#SET ARRAY TO GET EVENT_REQINFO TO FIND PROCESS NAME AND THE ROUTER NAME array set info [event_reqinfo] set process $info(process_name) set id $info(instance) set routername [info host]	Information supplied by **event_reqinfo** for TCL scripts is provided for the applets. With applets, the information is embedded in Cisco IOS Software, so arrays and variables do not need to be set up. Global variables such as the router name are not in the **event_reqinfo** information or a standard variable for applets; consequently, for both the TCL script and applet, the router name needs to be obtained independently.
action 2.0 Syslog msg "The Process ID $_process_instance ($_process_process_name) has restarted on this switch ($_info_routername)"	# SYSLOG MESSAGE action_Syslog msg "The Process ID $id ($process) has restarted on this switch ($routername)"	These equivalent commands invoke a syslog message.
action 3.0 mail server email.abc.com to IT_Ops@abc.com from $_info_routername@abc.com subject "The Process ID $_process_instance ($_process_process_name) has restarted on $_info_routername" body "This message is generated by EEM. Process $_process_process Restarted on $_info_routername. "	# SEND AN EMAIL # EMAIL MESSAGE set email_message "Mailservername: $_email_server From: $routername@abc.com To: IT_Ops@abc.com Cc: Subject: The Process ID $id ($process) has restarted on $routername This email is generated by EEM Process $process Restarted on $routername.	These equivalent commands send an email message. Notice that variables are used extensively in the email message. This approach increases the portability of both applets and TCL scripts. Note that the applet email body is slightly different because of the maximum number of characters per line in Cisco IOS Software. This maximum is a limitation in the use of applets rather than TCL. If the domain is left out of the "From" line in the email, the system will append the domain configured on the switch. Also, note that the domain—cat6k(config)#*ip domain name* (*name*)—needs to be configured on the switch for SMTP to work and the email to be sent.

	”
	
	# SYSLOG MESSAGE NOTIFICATION ON WHETHER THE EMAIL WAS SENT SUCCESSFULLY
	
	if {[catch {smtp_send_email $email_message} result]} {
	set result “Email send failed”
	} else {
	set result “Email Sent”
	}
	action_Syslog msg “$result”	