Embedded Syslog Manager (ESM)

The Embedded Syslog Manager (ESM) feature provides a programmable framework that allows you to filter, escalate, correlate, route, and customize system logging messages prior to delivery by the system message logger.

Restrictions for Embedded Syslog Manager

Embedded Syslog Manager (ESM) depends on the Tcl 8.3.4 Cisco IOS XE subsystem, because ESM filters are written in Tool Command Language (Tcl). ESM is available only in images that support Tcl version 8.3.4 or later versions. Support for Tcl 8.3.4 is added depending on your release.

ESM filters are written in Tcl.

ESM filtering cannot be applied to SNMP “history﷯? logging. Therefore, ESM filtering will not be applied to messages logged using the **logging history** and **snmp-server enable traps syslog** commands.

Information About the Embedded Syslog Manager

System Message Logging

With the introduction of the Embedded Syslog Manager, system messages can be logged independently as standard messages, XML-formatted messages, or ESM filtered messages. These outputs can be sent to any of the traditional syslog targets. For example, you could enable standard logging to the console connection, XML-formatted message logging to the buffer, and ESM filtered message logging to the monitor. Similarly, each type of output could be sent to different remote hosts. A benefit of separate logging processes is that standard logging will not be affected if, for example, there is some problem with the ESM filter modules.

System Logging Message Formatting

System logging messages are displayed in the following format:

%<facility>-<severity>-<mnemonic>: <message-text>

The following is an example of a system logging message:

```
%LINK-5-CHANGED: Interface Serial3/3, changed state to administratively down
```

Usually, these messages are preceded by additional text, such as the error sequence number and time stamp:

<sequence-number>: <time stamp>:%<facility>-<severity>-<mnemonic>: <message-text>

The following is an example of a system logging message preceded by an error sequence number and time stamp:

```
000013: Mar 18 14:52:10.039:%LINK-5-CHANGED: Interface Serial3/3, changed state
to administratively down
```

Note The time stamp format used in system logging messages is determined by the **service timestamps** global configuration mode command. The **service sequence-numbers** global configuration command enables or disables the leading sequence number. An asterisk (*) before the time indicates that the time may be incorrect because the system clock has not synchronized to a reliable time source.

Benefits of Embedded Syslog Manager

The Embedded Syslog Manager (ESM) is a feature integrated in Cisco software that allows complete control over system message logging at the source. ESM provides a programmatic interface to allow you to write custom filters that meet your specific needs relating to system logging. Benefits of this feature are:

- Customization--Fully customizable processing of system logging messages, with support for multiple, interfacing syslog collectors.
- Severity escalation for key messages--The ability to configure your own severity levels for syslog messages instead of using the system-defined severity levels.
- Specific message targeting--The ability to route specific messages or message types, based on type of facility or type of severity, to different syslog collectors.
- SMTP-base e-mail alerts--Capability for notifications using TCP to external servers, such as TCP-based syslog collectors or Simple Mail Transfer Protocol (SMTP) servers.
- Message limiting--The ability to limit and manage syslog “message storms” by correlating device-level events.

The ESM is not a replacement for the UDP-based syslog mechanism; instead, it is an optional subsystem that can operate in parallel with the current system logging process. For example, you can continue to have the original syslog message stream collected by server A, while the filtered, correlated, or otherwise customized ESM logging stream is sent to server B. All of the current targets for syslog messages (console, monitor,

buffer, and syslog host list) can be configured to receive either the original syslog stream or the ESM stream. The ESM stream can be further divided into user-defined streams and routed to collectors accordingly.

Syslog Filter Modules

Embedded Syslog Manager (ESM) uses syslog filter modules to process system logging messages. Syslog filter modules are scripts written in the Tool Command Language (Tcl) stored in local system memory or on a remote file server. The ESM is customizable because you can write and reference your own scripts.

Syslog filter modules can be written and stored as plain-text files or as precompiled files. Tcl script pre-compiling can be done with tools such as TclPro. Precompiled scripts allow a measure of security and managed consistency because they cannot be edited.

Note Because Tcl script modules contain executable commands, you should manage the security of these files using the same processes you use to manage configuration files.

How to Use the Embedded Syslog Manager

Writing ESM Syslog Filter Modules

Before referencing syslog filter modules in the Embedded Syslog Manager (ESM) configuration, you must write or obtain the modules you want to apply to system logging messages. Syslog filter modules can be stored in local system memory, or on a remote file server. Before you write syslog filter modules, you should understand the following concepts:

ESM Filter Process

When ESM is enabled, all system logging messages are processed through the referenced syslog filter modules. Syslog filter modules are processed in their order in the filter chain. The position of a syslog filter module in the filter chain is determined by the position tag applied in the **logging filter** global configuration mode command. If a position is not specified, the modules are processed in the order in which they were added to the configuration.

The output of each filter module is used as the input for the next filter module in the chain. Therefore, the Tcl global variable containing the original syslog message (::orig_msg) is set to the return value of each filter before invoking the next filter in the chain. Thus, if a filter returns NULL, no message will be sent out to the ESM stream. Once all filters have processed the message, the message is queued for distribution by the logger.

The console, buffer, monitor, and syslog hosts can be configured to receive a particular message stream (normal, XML, or ESM). The syslog hosts can be further restricted to receive user-defined numbered streams. Each target examines each message and accepts or rejects the message based on its stream tag. ESM filters can change the destination stream by altering the messages' stream tag by changing the Tcl global variable "::stream."

Syslog Filter Module Input

When Embedded Syslog Manager (ESM) is enabled, system logging messages are sent to the logging process. Each data element in the system logging message, and in the formatted syslog message as a whole, is recorded as a Tcl global variable. The data elements format for the syslog message are as follows:

<sequence-number>: <time stamp>:%<facility>-<severity>-<mnemonic>: <message-text>

The message-text will often contain message arguments.

When messages are received on a syslog host a “syslog-count” number is also added:

<syslog-count>: <sequence-number>: <time stamp>:%<facility>-<severity>-<mnemonic>: <message-text>

The following examples shows the syslog-count number included in the beginning of the sequence:

The table below lists the Tcl script input variables used in syslog filter modules. The syslog message data that the filter must operate on is passed as Tcl global namespace variables. Therefore, variables should be prefixed by a double-colon within the script module.

Standard ESM Filter Processing

Each time a system logging message is generated, the syslog filter modules are called in a series. This series is determined by the ::module_position variable, which in turn is typically the order in which the modules are referenced in the system configuration (the order in which they are configured).

The output of one filter module becomes the input to the next. Because the input to the filters is the Tcl global namespace variables, each filter can change any or all of these variables depending upon the purpose of the filter.

The only Tcl global variables that are automatically updated by the Embedded Syslog Manager (ESM) framework between subsequent filter executions are the ::orig_msg and ::cli_args variables. The framework automatically sets the value of ::orig_msg to the return value of the filter module. Thus a filter that is designed to alter or filter the original message must not manually set the value for the ::orig_msg variable; the filter only needs to return the desired value. For example, the following one-line ESM filter

```
return “This is my new syslog message.”
```

would ignore any message passed to it, and always change the output to the constant string “This is my new syslog message.” If the module was the last filter in the chain, all ESM targets would receive this string as the final syslog message.

The one-line ESM filter

```
return “”
```

would block all syslog messages to the ESM stream. For example, the line

```
return $::orig_msg
```

would do nothing but pass the message along to the next filter in the chain. Thus, an ESM filter designed to suppress unwanted messages would look something like this:

```
if { [my_procedure_to_check_this_message] == 1 } {
     return $::orig_msg
```

```
} else {
     return ""
}
```

Depending upon their design, some filters may not use the ::orig_msg variable at all, but rather reconstruct a syslog message from its data elements (using ::format_string, ::msg_args, ::timestamp, and so on). For example, an XML tagging filter will tag the individual data elements, and disregard the original formatted message. It is important for such modules to check the ::orig_msg variable at the beginning of the Tcl script, so that if a previous filter indicated that the message should not be sent out (::orig_msg is NULL), the message would not be processed, but return NULL also.

Commands can also be added to syslog filter modules using the **exec** and **config** Tcl commands. For example, if you wanted to add the source IP address to the syslog messages, and syslog messages were configured to be sent from the Ethernet 2/0 interface (using the **logging source-interface** command) you could issue the **show interface Ethernet 2/0** command during the module initialization by using the **exec** Tcl command within the script:

```
set source_ip_string [exec show ip int E2/0 | inc Internet]
puts $source_ip_string
"  Internet address is 10.4.2.63/24"
```

Background ESM Filter Processing

In Tcl, commands can be queued for future processing by using the **after** Tcl command. The most common use of this command is to correlate (gather and summarize) events over a fixed interval of time, called the "correlation window." Once the window of interest expires, the filter will need to "wake up," and calculate or summarize the events that occurred during the window, and often send out a new syslog message to report the events. This background process is handled by the ESM Event Loop process, which allows the Tcl interpreter to execute queued commands after a certain amount of time has passed.

If your syslog filter module needs to take advantage of correlation windows, it must use the **after** Tcl command to call a summary procedure once the correlation window expires (see examples in the "Configuration Examples for the Embedded Syslog Manager" section). Because there is no normal filter chain processing when background processes are run, in order to produce output these filters must use one of two ESM Tcl extensions: **errmsg** or **esm_errmsg**.

During background processing, the commands that have been queued by the **after** command are not run in the context of the filter chain (as in normal processing), but rather are autonomous procedures that are executed in series by the Tcl interpreter. Thus, these background procedures should not operate on the normal Tcl global namespace variables (except for setting the global namespace variables for the next filter when using **esm_errmsg**), but should operate on variables stored in their own namespace. If these variables are declared outside of a procedure definition, they will be persistent from call to call.

The purpose of the **errmsg** Tcl command is to create a new message and send it out for distribution, bypassing any other syslog filter modules. The syntax of the **errmsg** command is:

```
errmsg <severity> <stream> <message_string>
```

The purpose of the **esm_errmsg** Tcl command is to create a new message, process it with any syslog filter modules below it in the filter chain, and then send it out for distribution. The syntax of the **esm_errmsg** command is:

```
esm_errmsg <module_position>
```

The key difference between the errmsg() Tcl function and the esm_errmsg() Tcl function is that **errmsg** ignores the filters and directly queues a message for distribution, while **esm_errmsg** will send a syslog message down the chain of filters.

In the following example, a new syslog message is created and sent out tagged as Alert severity 1 to the configured ESM logging targets (stream 2). The purpose of this filter is to suppress the individual SYS-5-CONFIG messages over a thirty minute correlation window, and send out a summary message at the end of the window.

```
errmsg 1 2 "*Jan 24 09:34:02.539: %SYS-1-CONFIG_I: There have been 12
configuration changes to the router between Jan 24 09:04:02.539 and Jan 24
09:34:01.324"
```

In order to use **esm_errmsg**, because the remaining filters following this one will be called, this background process must populate the needed Tool Command Language (Tcl) global namespace variables prior to calling **esm_errmsg**. Passing the ::module_position tells the ESM framework which filter to start with. Thus, filters using the **esm_errmsg** command should store their ::module_position (passed in the global namespace variables during normal processing) in their own namespace variable for use in background processing. Here is an example:

```
proc ::my_filter_namespace::my_summary_procedure{}
{
 set ::orig_msg "*Jan 24 09:34:02.539: %SYS-1-CONFIG_I: There have been 12
configuration changes to the router between Jan 24 09:04:02.539 and Jan 24
09:34:01.324"
 set ::timestamp "*Jan 24 09:34:02.539"
 set ::severity 1
 set ::stream 2
 set ::traceback ""
 set ::pid ""
set ::process ""
set ::format_string "There have been %d configuration changes to the router
between %s and %s"
set ::msg_args {12 "Jan 24 09:04:01.539" "Jan 24 09:34:01.324"}
esm_errmsg $::my_filter_namespace::my_module_position
}
```

The benefit of setting all the global namespace variables for the **esm_errmsg** command is that your filters will be modular, and the order they are used in the ESM framework will not matter. For example, if you want all of the messages destined for the ESM targets to be suffixed with the message originator's hostname, you could write a one-line "hostname" filter and place it at the bottom of the filter chain:

```
return "$::orig_msg -- $::hostname"
```

In this example, if any of your filters generate new messages during background processing and they use **esm_errmsg** instead of **errmsg**, these messages will be clearly suffixed with the hostname.

What to Do Next

After creating your syslog filter module, you should store the file in a location accessible to the device. You can copy the file to local system memory, or store it on a network file server.

Configuring the Embedded Syslog Manager

To configure the Embedded Syslog Manager (ESM), specify one or more filters to be applied to generated syslog messages, and specify the syslog message target.

Before you begin

One or more syslog filter modules must be available to the device.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **logging filter** *filter-url* [*position*] [**args** *filter-arguments*]
4. Repeat Step 3 for each syslog filter module that should be applied to system logging output.
5. Enter one of the following:
   - **logging** [**console** | **buffered** | **monitor**] **filtered** [*security-level*]
   - or
   - **logging host** {*ip-address* | *hostname*} **filtered** [**stream** *stream-id*]
6. Repeat Step 5 for each desired system logging destination.
7. **logging source-interface** *type number*
8. **logging origin-id** {**hostname** | **ip** | **ipv6** |**string** *user-defined-id*}
9. **end**
10. **show logging**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**logging filter** *filter-url* [*position*] [**args** *filter-arguments*] **Example:** `Device(config)# logging filter slot0:/escalate.tcl 1 args CONFIG_I 1`	Specifies one or more syslog filter modules to be applied to generated system logging messages. • Repeat this command for each syslog filter module that should be used. • The *filter-url* argument is the Cisco IOS File System location of the syslog filter module (script). The location can be in local memory, or a remote server using **tftp:**, **ftp:**, or **rcp:**.

	Command or Action	Purpose
		• The optional *position* argument specifies the order in which the syslog filter modules should be executed. If this argument is omitted, the specified module will be positioned as the last module in the chain. • Filters can be reordered quickly by again entering the **logging filter** command and specifying a different position. • The optional **args** *filter-arguments* syntax can be added to pass arguments to the specified filter. Multiple arguments can be specified. The number and type of arguments should be defined in the syslog filter module. For example, if the syslog filter module is designed to accept a specific e-mail address as an argument, you could pass the e-mail address using the **args user@host.com** syntax. Multiple arguments are typically delimited by spaces. • To remove a module from the list of modules to be executed, use the **no** form of this command.
Step 4	Repeat Step 3 for each syslog filter module that should be applied to system logging output.	--

Step 5	Enter one of the following:
	• **logging** [**console** | **buffered** | **monitor**] **filtered** [*security-level*]
	• or
	• **logging host** {*ip-address* | *hostname*} **filtered** [**stream** *stream-id*]
	Example:
	`Device(config)# logging console filtered informational`
	Example:
	`Device(config)# logging host 209.165.200.225 filtered stream 20`
		Specifies the target for ESM filtered syslog output.
		• ESM filtered syslog messages can be sent to the console, a monitor (TTY and Telnet connections), the system buffer, or remote hosts.
		• The optional *level* argument limits the sending of messages to those at or numerically lower than the specified value. For example, if level **1** is specified, only messages at level 1 (alerts) or level 0 (emergencies) will be sent to the specified target. The level can be specified as a keyword or number.
		• When you log to the console, monitor connection, or system buffer, the severity threshold specified by the *level* argument takes precedence over the ESM filtering. Even if the ESM filters return a message to be delivered to ESM targets, if the severity does not meet the configured threshold (is numerically higher than the level value), the message will not be delivered.
		• When you log to remote hosts, the stream tag allows you to specify a destination based on the type of message. The **stream** *stream-id* syntax allows you to configure the ESM to send only messages that have a specified stream value to a certain host.

	Command or Action	Purpose
		• The stream value is applied to messages by the configured syslog filter modules. For example, all Severity 5 messages could have a stream tag of "20" applied. You can then specify that all messages with a stream tag of "20" be sent to the host at 209.165.200.225.:
Step 6	Repeat Step 5 for each desired system logging destination.	• By issuing the logging host command multiple times, you can specify different targets for different system logging streams. • You can configure messages at different severity levels to be sent to the console, monitor connection, or system buffer. For example, you may want to display only important messages to the screen (using a monitor or console connection) at your network operations center (NOC).
Step 7	**logging source-interface** *type number* **Example:** `Device(config)# logging source-interface Gigabitethernet 0/0`	(Optional) Specifies the source interface for syslog messages sent to remote syslog hosts. • Normally, a syslog messages sent to remote hosts will use whatever interface is available at the time of the message generation. This command forces the device to send syslog messages to remote hosts only from the specified interface.
Step 8	**logging origin-id** {**hostname** | **ip** | **ipv6** |**string** *user-defined-id*} **Example:** `Device(config)# logging origin-id string "Domain 2, Router 5"`	(Optional) Allows you to add an origin identifier to syslog messages sent to remote hosts. • The origin identifier is added to the beginning of all syslog messages sent to remote hosts. The identifier can be the hostname, the IP address, or any text that you specify. • The origin identifier is useful for identifying the source of system logging messages in cases where you send syslog output from multiple devices to a single syslog host.
Step 9	**end** **Example:** `Device(config)# end`	Ends your current configuration session and returns the CLI to privileged EXEC mode.
Step 10	**show logging** **Example:** `Device# show logging`	(Optional) Displays the status of system logging, including the status of ESM filtered logging. • If filtered logging to the buffer is enabled, this command also shows the data stored in the buffer.

	Command or Action	Purpose
		• The order in which syslog filter modules are listed in the output of this command is the order in which the filter modules are executed.

Configuration Examples for the Embedded Syslog Manager

Example: Configuring the Embedded Syslog Manager Example

In the following example, the Embedded Syslog Manager (ESM) filter logging is enabled for the console connection, standard logging is enabled for the monitor connection and for the buffer, and XML-formatted logging is enabled for the host at 209.165.200.225:

```
Device(config)# logging filter tftp://209.165.200.225/ESM/escalate.tcl
Device(config)# logging filter slot0:/email.tcl user@example.com
Device(config)# logging filter slot0:/email_guts.tcl
Device(config)# logging console filtered
Device(config)# logging monitor 4
Device(config)# logging buffered debugging
Device(config)# logging host 209.165.200.225 xml
Device(config)# end

Device# show logging
Syslog logging: enabled (0 messages dropped, 8 messages rate-limited,
                0 flushes, 0 overruns, xml disabled, filtering enabled)
    Console logging: level debugging, 21 messages logged, xml disabled,
                     filtering enabled
    Monitor logging: level warnings , 0 messages logged, xml disabled,
                     filtering disabled
    Buffer logging: level debugging, 30 messages logged, xml disabled,
                     filtering disabled
    Logging Exception size (8192 bytes)
    Count and timestamp logging messages: disabled

Filter modules:
    tftp://209.165.200.225/ESM/escalate.tcl
    slot0:/email.tcl user@example.com

    Trap logging: level informational, 0 message lines logged
        Logging to 209.165.200.225, 0 message lines logged, xml enabled,
              filtering disabled

Log Buffer (8192 bytes):

*Jan 24 09:34:28.431: %SYS-5-CONFIG_I: Configured from console by console
*Jan 24 09:34:51.555: %SYS-5-CONFIG_I: Configured from console by console
*Jan 24 09:49:44.295: %SYS-5-CONFIG_I: Configured from console by console
Device#
```

Example: Syslog Filter Module

Syslog Script Modules are Tcl scripts. The following examples are provided to assist you in developing your own Syslog Script Modules.

Note These script modules are provided as examples only, and are not supported by Cisco. No guarantees, expressed or implied, are provided for the functionality or impact of these scripts.

Example: Severity Escalation

This ESM syslog filter module example watches for a single mnemonic (supplied via the first CLI argument) and escalates the severity of the message to that specified by the second CLI argument.

```
#  ====================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  Severity Escalation Filter                          ||||      ||||
#                                                   ..:||||||:..:||||||:..
#                                                  ------------------------
#                                                  C i s c o  S y s t e m s
#  ====================================================================
#
# Usage: Set CLI Args to "mnemonic new_severity"
#
# Namespace: global
# Check for null message
if { [string length $::orig_msg] == 0} {
   return ""
}

if { [info exists ::cli_args] } {
    set args [split $::cli_args]
    if { [ string compare -nocase [lindex $args 0] $::mnemonic ] == 0 } {
        set ::severity [lindex $args 1]
        set sev_index [ string first [lindex $args 0] $::orig_msg ]
        if {  $sev_index >= 2 } {
           incr sev_index -2
           return [string replace $::orig_msg $sev_index $sev_index \
              [lindex $args 1]]
        }
    }
}
return $::orig_msg
```

Example: Message Counting

This ESM syslog filter module example is divided into two files for readability. The first file allows the user to configure those messages that they want to count and how often to summarize (correlation window) by populating the msg_to_watch array. The actual procedures are in the counting_guts.tcl file. Note the use of the separate namespace “counting” to avoid conflict with other ESM filters that may also perform background processing.

```
#  ====================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
```

```
#  Message Counting Filter                          ||||      ||||
#                                                ..:||||||:..:||||||:..
#                                               -----------------------
#                                               C i s c o  S y s t e m s
# ============================================================================


#
# Usage:
# 1) Define the location for the counting_guts.tcl script
#
# 2) Define message categories to count and how often to dump them (sec)
#    by populating the "msg_to_watch" array below.
#    Here we define category as facility-severity-mnemonic
#    Change dump time to 0 to disable counting for that category
#
# Namespace: counting
namespace eval ::counting {
    set sub_script_url tftp://172.16.0.0/12/ESM/counting_guts.tcl
    array set msg_to_watch {
        SYS-5-CONFIG_I            5
    }
#  ====================== End User Setup ==============================
# Initialize processes for counting
    if { [info exists init] == 0 } {
       source $sub_script_url
       set position $module_position
    }
# Process the message
process_category
} ;# end namespace counting
```

Message Counting Support Module (counting_guts.tcl)

```
#  ====================================================================
#  Embedded Syslog Manager                            ||        ||
#                                                     ||        ||
#  Message Counting Support Module                   ||||      ||||
#                                                 ..:||||||:..:||||||:..
#   (No User Modification)                       -----------------------
#                                                C i s c o  S y s t e m s
# ============================================================================



namespace eval ::counting {

# namespace variables

array set cat_msg_sev {}
array set cat_msg_traceback {}
array set cat_msg_pid {}
array set cat_msg_proc {}
array set cat_msg_ts {}
array set cat_msg_buginfseq {}
array set cat_msg_name {}
array set cat_msg_fac {}
array set cat_msg_format {}
array set cat_msg_args {}
array set cat_msg_count {}
array set cat_msg_dump_ts {}
```

```
# Should I count this message ?
   proc query_category {cat} {
        variable msg_to_watch
        if { [info exists msg_to_watch($cat)] } {
            return $msg_to_watch($cat)
        } else {
            return 0
        }
    }
   proc clear_category {index} {
        variable cat_msg_sev
        variable cat_msg_traceback
        variable cat_msg_pid
        variable cat_msg_proc
        variable cat_msg_ts
        variable cat_msg_buginfseq
        variable cat_msg_name
        variable cat_msg_fac
        variable cat_msg_format
        variable cat_msg_args
        variable cat_msg_count
        variable cat_msg_dump_ts
        unset cat_msg_sev($index) cat_msg_traceback($index) cat_msg_pid($index)\
              cat_msg_proc($index) cat_msg_ts($index) \
              cat_msg_buginfseq($index)  cat_msg_name($index) \
              cat_msg_fac($index) cat_msg_format($index) cat_msg_args($index)\
              cat_msg_count($index) cat_msg_dump_ts($index)
    }
# send out the counted messages
   proc dump_category {category} {
        variable cat_msg_sev
        variable cat_msg_traceback
        variable cat_msg_pid
        variable cat_msg_proc
        variable cat_msg_ts
        variable cat_msg_buginfseq
        variable cat_msg_name
        variable cat_msg_fac
        variable cat_msg_format
        variable cat_msg_args
        variable cat_msg_count
        variable cat_msg_dump_ts
        variable poll_interval
        set dump_timestamp [cisco_service_timestamp]
foreach index [array names cat_msg_count $category] {
      set fsm "$cat_msg_fac($index)-$cat_msg_sev($index)-$cat_msg_name($index)"
      set ::orig_msg \
        [format "%s%s: %%%s: %s %s %s %s - (%d occurence(s) between %s and %s)"\
        $cat_msg_buginfseq($index)\
          $dump_timestamp\
          $fsm \
          [uplevel 1 [linsert $cat_msg_args($index) 0 ::format
$cat_msg_format($index) ]] \
          $cat_msg_pid($index) \
          $cat_msg_proc($index) \
          $cat_msg_traceback($index) \
          $cat_msg_count($index) \
          $cat_msg_ts($index) \
          $dump_timestamp]
# Prepare for remaining ESM filters
               set ::severity $cat_msg_sev($index)
               set ::traceback $cat_msg_traceback($index)
               set ::pid $cat_msg_pid($index)
```

```
                set ::process $cat_msg_proc($index)
                set ::timestamp $cat_msg_ts($index)
                set ::buginfseq $cat_msg_buginfseq($index)
                set ::mnemonic $cat_msg_name($index)
                set ::facility $cat_msg_fac($index)
                set ::format_string $cat_msg_format($index)
                set ::msg_args [split $cat_msg_args($index)]
                esm_errmsg $counting::position
                clear_category $index
        }
  }
# See if this message already has come through since the last dump.
# If so, increment the count, otherwise store it.
   proc process_category {} {
        variable cat_msg_sev
        variable cat_msg_traceback
        variable cat_msg_pid
        variable cat_msg_proc
        variable cat_msg_ts
        variable cat_msg_buginfseq
        variable cat_msg_name
        variable cat_msg_fac
        variable cat_msg_format
        variable cat_msg_args
        variable cat_msg_count
        variable cat_msg_dump_ts
        if { [string length $::orig_msg] == 0} {
           return ""
        }
        set category "$::facility-$::severity-$::mnemonic"
        set correlation_window [expr [ query_category $category ] * 1000]
        if { $correlation_window == 0 } {
            return $::orig_msg
        }
        set message_args [join $::msg_args]
        set index "$category,[lindex $::msg_args 0]"
        if { [info exists cat_msg_count($index)] } {
                 incr cat_msg_count($index)
         } else {
              set cat_msg_sev($index) $::severity
              set cat_msg_traceback($index) $::traceback
              set cat_msg_pid($index) $::pid
              set cat_msg_proc($index) $::process
              set cat_msg_ts($index) $::timestamp
              set cat_msg_buginfseq($index) $::buginfseq
              set cat_msg_name($index) $::mnemonic
              set cat_msg_fac($index) $::facility
              set cat_msg_format($index) $::format_string
              set cat_msg_args($index) $message_args
              set cat_msg_count($index) 1
              set cat_msg_dump_ts($index) [clock seconds]
              catch [after $correlation_window counting::dump_category $index]
         }
         return ""
   }
# Initialized
set init 1
} ;#end namespace counting
```

Example: XML Tagging

This ESM syslog filter module applies user-defined XML tags to syslog messages:

```
#  ========================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  XML Tagging Filter                                  ||||      ||||
#                                                   ..:||||||:..:||||||:..
#                                                  -----------------------
#                                                  C i s c o  S y s t e m s
#  ========================================================================
#
# Usage: Define desired tags below.
#
# Namespace: xml
# Check for null message
    if { [string length $::orig_msg] == 0} {
       return ""
    }
namespace eval xml {
####  define tags ####
set MSG_OPEN "<ios-log-msg>"
set MSG_CLOSE "</ios-log-msg>"
set FAC_OPEN   "<facility>"
set FAC_CLOSE  "</facility>"
set SEV_OPEN   "<severity>"
set SEV_CLOSE  "</severity>"
set MNE_OPEN   "<msg-id>"
set MNE_CLOSE  "</msg-id>"
set SEQ_OPEN   "<seq>"
set SEQ_CLOSE  "</seq>"
set TIME_OPEN  "<time>"
set TIME_CLOSE "</time>"
set ARGS_OPEN  "<args>"
set ARGS_CLOSE "</args>"
set ARG_ID_OPEN "<arg id="
set ARG_ID_CLOSE "</arg>"
set PROC_OPEN "<proc>"
set PROC_CLOSE "</proc>"
set PID_OPEN "<pid>"
set PID_CLOSE "</pid>"
set TRACE_OPEN "<trace>"
set TRACE_CLOSE "</trace>"
#  ======================= End User Setup ==============================
#### clear result ####
set result ""
#### message opening, facility, severity, and name ####
append result $MSG_OPEN $FAC_OPEN $::facility $FAC_CLOSE $SEV_OPEN $::severity
$SEV_CLOSE $MNE_OPEN $::mnemonic $MNE_CLOSE
#### buginf sequence numbers ####
if { [string length $::buginfseq ] > 0 } {
    append result $SEQ_OPEN $::buginfseq $SEQ_CLOSE
}
#### timestamps ####
if { [string length $::timestamp ] > 0 } {
    append result $TIME_OPEN $::timestamp $TIME_CLOSE
}
#### message args ####
if { [info exists ::msg_args] } {
    if { [llength ::msg_args] > 0 } {
        set i 0
        append result $ARGS_OPEN
        foreach arg $::msg_args {
            append result $ARG_ID_OPEN $i ">" $arg $ARG_ID_CLOSE
                incr i
        }
```

```
        append result $ARGS_CLOSE
    }
}
#### traceback ####
if { [string length $::traceback ] > 0 } {
    append result $TRACE_OPEN $::traceback $TRACE_CLOSE
}
#### process ####
if { [string length $::process ] > 0 } {
    append result $PROC_OPEN $::process $PROC_CLOSE
}
#### pid ####
if { [string length $::pid ] > 0 } {
    append result $PID_OPEN $::pid $PID_CLOSE
}
#### message close ####
append result $MSG_CLOSE
return "$result"
} ;# end namespace xml
```

Example: SMTP-Based E-Mail Alert

This ESM syslog filter module example watches for configuration messages and sends them to the e-mail address supplied as a CLI argument. This filter is divided into two files. The first file implements the filter, and the second file implements the Simple Mail Transfer Protocol (SMTP) client.

```
#  ====================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  Email Filter                                        ||||      ||||
#  (Configuration Change Warning)                 ..:||||||:..:||||||:..
#                                                ------------------------
#                                                 C i s c o  S y s t e m s
#   ====================================================================
# Usage:  Provide email address as CLI argument.  Set email server IP in
#         email_guts.tcl
#
# Namespace: email
if { [info exists email::init] == 0 } {
   source tftp://123.123.123.123/ESM/email_guts.tcl
}
# Check for null message
if { [string length $::orig_msg] == 0} {
      return ""
   }
if { [info exists ::msg_args] } {
    if { [string compare -nocase CONFIG_I $::mnemonic ] == 0 } {
                email::sendmessage $::cli_args $::mnemonic \
                [string trim $::orig_msg]
    }
}
return $::orig_msg
```

E-Mail Support Module (email_guts.tcl)

```
#  ====================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  Email Support Module                                ||||      ||||
#                                                   ..:||||||:..:||||||:..
#                                                  ------------------------
```

```
#                                                 C i s c o  S y s t e m s
#   ================================================================
#
# Usage: Set email host IP, from, and friendly strings below.
#
namespace eval email {
    set sendmail(smtphost)172.16.0.1
    set sendmail(from) $::hostname
    set sendmail(friendly) $::hostname
    proc sendmessage {toList subject body} {
        variable sendmail
        set smtphost $sendmail(smtphost)
        set from $sendmail(from)
        set friendly $sendmail(friendly)
        set sockid [socket $smtphost 25]
## DEBUG
set status [catch {
        puts $sockid "HELO $smtphost"
        flush $sockid
        set result [gets $sockid]
        puts $sockid "MAIL From:<$from>"
        flush $sockid
        set result [gets $sockid]
        foreach to $toList {
            puts $sockid "RCPT To:<$to>"
            flush $sockid
        }
        set result [gets $sockid]
        puts  $sockid "DATA "
        flush $sockid
        set result [gets  $sockid]
        puts  $sockid "From: $friendly <$from>"
        foreach to $toList {
            puts $sockid "To:<$to>"
        }
        puts  $sockid "Subject: $subject"
        puts  $sockid "\n"
        foreach line [split $body "\n"] {
            puts  $sockid " $line"
        }
        puts  $sockid "."
        puts  $sockid "QUIT"
        flush $sockid
        set result [gets  $sockid]
} result]
        catch {close $sockid }
    if {$status} then {
        return -code error $result
    }
}
} ;# end namespace email
set email::init 1
```

Example: Stream

This ESM syslog filter module example watches for a given facility (first CLI argument) and routes these messages to a given stream (second CLI argument:.

```
#  ===================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  Stream Filter (Facility)                            ||||      ||||
#                                                  ..:||||||:..:||||||:..
```

```
#                                                 -----------------------
#                                                  C i s c o  S y s t e m s
#   ================================================================
# Usage:  Provide facility and stream as CLI arguments.
#
# Namespace: global
# Check for null message
#  ====================== End User Setup ==============================
set args [split $::cli_args]
if { [info exists ::msg_args] } {
    if { $::facility == [lindex $args 0] } {
                set ::stream [lindex $args 1]
    }
}
return $::orig_msg}
```

Example: Source IP Tagging

The **logging source-interface** CLI command can be used to specify a source IP address in all syslog packets sent from the device. The following syslog filter module example demonstrates the use of **show** CLI commands (**show running-config** and **show ip interface** in this case) within a filter module to add the source IP address to syslog messages. The script looks for the local namespace variable “source_ip::init” first. If the variable is not defined in the first syslog message processed, the filter will run the **show** commands and use regular expressions to get the source interface and then its IP address.

Note that in this script, the **show** commands are run only once. If the source interface or its IP address were to be changed, the filter would have to be reinitialized to pick up the new information. (You could have the show commands run on every syslog message, but this would not scale well.)

```
#  ====================================================================
#  Embedded Syslog Manager                              ||        ||
#                                                       ||        ||
#  Source IP Module                                    ||||      ||||
#                                                   ..:||||||:..:||||||:..
#                                                  -----------------------
#                                                   C i s c o  S y s t e m s
#   ================================================================
# Usage: Adds Logging Source Interface IP address to all messages.
#
# Namespace:source_ip
#
#  ====================== End User Setup ==============================
namespace eval ::source_ip {
    if { [info exists init] == 0 } {
       if { [catch {regexp {^logging source-interface (.*$)} [exec show
run | inc logging source-interface] match source_int}]} {
            set suffix "No source interface specified"
       } elseif { [catch {regexp {Internet address is (.*)/.*$} [exec
show ip int $source_int | inc Internet] match ip_addr}]} {
            set suffix "No IP address configured for source interface"
       } else {
            set suffix $ip_addr
       }
       set init 1
    }

    if { [string length $::orig_msg] == 0} {
      return ""
    }
    return "$::orig_msg - $suffix"
} ;# end namespace source_ip
```

Additional References for the Embedded Syslog Manager

Related Documents

Related Topic	Document Title
Cisco IOS XE Commands	Command Lookup Tool
System Message Logging	Troubleshooting and Fault Management module
XML Formatted System Message Logging	XML Interface to Syslog Messages module
Tcl 8.3.4 Support in Cisco Software	*Cisco IOS Scripting with Tcl* module
Network Management commands (including logging commands): complete command syntax, defaults, command mode, command history, usage guidelines, and examples	*Cisco IOS Network Management Command Reference*

Standards and RFCs

Standard/RFC	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--
RFC-3164	*The BSD Syslog Protocol* This RFC is informational only. The Cisco implementation of syslog does not claim full compliance with the protocol guidelines mentioned in this RFC. Not all supported RFCs are listed.

MIBs

MIB	MIBs Link
No new or modified standards are supported, and support for existing standards has not been modified.	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

Technical Assistance

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

Feature Information for the Embedded Syslog Manager

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

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

Table 1: Feature Information for the Embedded Syslog Manager

Feature Name	Releases	Feature Information
Embedded Syslog Manager		The Embedded Syslog Manager (ESM) feature provides a programmable framework that allows you to filter, escalate, correlate, route, and customize system logging messages prior to delivery by the Cisco IOS system message logger.

Glossary

Note Refer to the "Internetworking Terms and Acronyms" section for terms not included in this glossary.

console--Specifies the connection (CTY or console line) to the console port of the device. Typically, this is a terminal attached directly to the console port, or a PC with a terminal emulation program. Corresponds to the **show terminal** command.

monitor--Specifies the TTY (TeleTYpe terminal) line connection at a line port. In other words, the “monitor” keyword corresponds to a terminal line connection or a Telnet (terminal emulation) connection. TTY lines (also called ports) communicate with peripheral devices such as terminals, modems, and serial printers. An example of a TTY connection is a PC with a terminal emulation program connected to the device using a dialup modem.

SEMs--Abbreviation for system error messages. “System error messages” is the term formerly used for messages generated by the system logging (syslog) process. Syslog messages use a standardized format, and come in eight severity levels, from “emergencies” (level 0) to “debugging” (level 7). The term “system error

message" is actually misleading, because these messages can include notifications of device activity beyond "errors" (such as informational notices).

syslog--Abbreviation for the system message logging process in Cisco software. Also used to identify the messages generated, as in "syslog messages." Technically, the term "syslog" refers only to the process of logging messages to a remote host or hosts, but is commonly used to refer to all Cisco system logging processes.

trap--A trigger in the system software for sending error messages. "Trap logging" means logging messages to a remote host. The remote host is actually a syslog host from the perspective of the device sending the trap messages, but because the receiving device typically provides collected syslog data to other devices, the receiving device is also referred to as a "syslog server."