Configuring CSM Scripts

This chapter describes how to configure content switching and contains these sections:

- Configuring TCL Scripts, page 10-1
- Configuring Scripts for Health Monitoring Probes, page 10-10

Configuring TCL Scripts

The CSM now allows you to upload and execute Toolkit Command Language (TCL) scripts on the CSM. TCL scripts allow you to write customized TCL scripts to develop customized health probes or standalone tasks. The TCL interpreter code in CSM is based on Release 8.0 of the standard TCL distribution. You can create a script to configure health probes (see "Configuring Probes for Health Monitoring" section on page 9-1) or perform tasks on the CSM that are not part of a health probe. The CSM periodically executes the scripts at user-configurable intervals.

Before CSM Release 3.1(1a), you could not configure a health probe for a protocol that did not include the basic health-monitoring code. You can now write probes to customize the CSM for your specific application. CSM Release 3.2 supports UDP socket functions. TCL is a widely used scripting language within the networking community. TCL also has huge libraries of scripts developed that can easily found from various sites.

The CSM currently supports two script modes:

- Health-monitoring script mode—These scripts must be written using some simple rules. The execution of these scripts is controlled by health-monitoring module.
- Standalone script mode—These scripts are TCL scripts that you can download from the web. You control the execution of these scripts through the CSM configuration.

For your convenience, sample scripts are available to support the TCL feature. Other custom scripts will work, but these sample scripts are supported by Cisco TAC. The file with sample scripts is located at this URL:

http://www.cisco.com/cgi-bin/tablebuild.pl/cat6000-intellother

The file containing the scripts is named: c6slb-script.3-1-1a.tcl.

These sections describe how to load prepared TCL scripts, write your own customized TCL scripts, and then run your TCL scripts:

- Loading Scripts, page 10-2
- Writing TCL Scripts, page 10-3
- Running TCL Scripts, page 10-8

Loading Scripts

Scripts are loaded onto the CSM through script files. A script file may contain zero, one, or more scripts. Each script requires 128 KB of stack space. Because there can be a maximum of 50 health scripts, the maximum stack space for script probes is 6.4 MB. Standalone scripts may also be running, which would consume more stack space.

Scripts can be loaded from a TFTP server, bootflash, slot0, and other storage devices using the **script file** [*file-url*] command.

This example shows how to load a script:

```
Router(config)# module csm 4
Router(config-module-csm)# script file tftp://192.168.1.1/httpProbe.test
```

The script name is either the filename of the script or a special name encoded within the script file. Each script file may contain a number scripts in the same file. To run the script or create a health probe using that script, you must refer to the script name, not the script file from which the script was loaded.

In order to identify each relevant script, each script must start with a line:

#!name = *script_name*

This example shows a master script file in which the scripts are bundled:

```
#!name = SCRIPT1
puts "this is script1"
!name = SCRIPT2
puts "this is script2"
```

This example shows how to find the scripts available in a master script file:

```
Router(config)# configure terminal
Router(config-t)# module csm 4
Router(config-module-csm)# script file tftp://192.168.1.1/script.master
Router(config-module-csm)# end
```

This example shows three scripts available from the script.master file:

```
Router(config)# show module csm 4 file tftp://192.168.1.1/script.master
RECURSIVE_TCL, file tftp://192.168.1.1/script.master
  size = 304, load time = 03:49:36 UTC 03/26/93
ECHO_TCL, file tftp://192.168.1.1/script.master
  size = 228, load time = 03:49:36 UTC 03/26/93
HTTP_PROBE, file tftp://192.168.1.1/script.master
  size = 920, load time = 03:49:36 UTC 03/26/93
```

To show the contents of a loaded script file, use this command:

```
Router(config)# show module csm slot script full_file_URL code
```

One major difference between a script task and a script probe is that the execution of a health script is scheduled by the health monitoring CSM module. These conditions apply:

- A script can be modified while a script probe is active. The changes are applied automatically in the next script execution, and for command line arguments.
- During probe configuration, a particular script is attached to the probe. If the script is unavailable at that time, the probe executes with a null script. If this situation occurs, a warning flag is generated. However, when the script is loaded again, the binding between the probe object and the script does not run automatically. You must use the **no script** and **script** commands again to do the binding.
- After a script is loaded it remains in the system and cannot be removed. You can modify a script by changing a script, and then by entering the **no script file** and **script file** commands again.

- Each script is always identified by its unique name. If two or more scripts have identical names, the last loaded script is used by the CSM. When there are duplicate script names, a warning message is generated by the CSM.

Writing TCL Scripts

The CSM Release 3.1(1a) TCL script feature is based on the TCL 8.0 source distribution software. CSM TCL is modified so that it can be interrupted to call another process unlike the standard TCL library, allowing for concurrent TCL interpreter execution. The CSM TCL library does not support any standard TCL file I/O command, such as file, fcopy, and others. Table 10-1 lists the TCL commands that are supported by CSM.

Table 10-1 TCL Commands Supported by the CSM

Command			
Generic TCL Commands			
append	array	binary	break
catch	concat	continue	error
eval	exit	expr	fblocked
for	foreach	format	global
gets	if	incr	info
join	lappend	lindex	linsert
list	llength	lrange	lreplace
lsearch	lsort	proc	puts
regexp	regsub	rename	return
set	split	string	subst
switch	unset	uplevel	upvar
variable	while	namespace	
Time-Related Commands			
after	clock	time	
Socket Commands			
close	blocked	fconfigured	fileevent
flush	eof	read	socket
update	vwait		
UDP Command Set			
udp bind handle readable [script]	udp bind handle writable [script]	udp connect host port	udp close handle
udp info handle	udp open [port]	udp receive handle	udp send handle [host port] message
CSM-Specific Commands			
ping	enable real	disable real	

Table 10-2 TCL Commands Not Supported by the CSM

Generic TCL Commands			
cd	fcopy	file	open
seek	source	tell	filename
load	package		

UDP Commands

The UDP command set allows Scotty based TCL scripts to run on the CSM. Scotty is the name of a software package that allows you to implement site-specific network management software using high-level, string-based APIs.

Table 10-3

Command	Definition
udp open [port]	Opens a UDP datagram socket and returns a UDP handle. The socket is bound to given port number or name. An unused port number is used if the port argument is missing.
udp connect host port	Opens a UDP datagram socket and connects it to a port on a remote host. A connected UDP socket only allows sending messages to a single destination. This usually allows shortening the code because there is no need to specify the destination address for each **udp send** command on a connected UDP socket. The command returns a UDP handle.
udp send handle [host port] message	Sends a datagram containing a message to the destination specified by host and port. The host and port arguments may not be used if the UDP handle is already connected to a transport endpoint. If the UDP handle is not connected, you must use these optional arguments to specify the destination of the datagram.
udp receive handle	Receives a datagram from the UDP socket associated with handle. This command blocks until a datagram is ready to be received. In most cases, it might be a good idea to check for pending datagrams using the **udp bind** command.
udp close handle	Closes the UDP socket associated with handle.
udp bind handle readable [script] udp bind handle writable [script]	Allows binding scripts to a UDP handle. A script is evaluated once the UDP handle becomes either readable or writable, depending on the third argument of the **udp bind** command. The script currently bound to a UDP handle can be retrieved by calling the **udp bind** command without a script argument. Bindings are removed by binding an empty string.
udp info [handle]	Without the handle argument this command returns a list of all existing UDP handles. Information about the state of a UDP handle can be obtained by supplying a valid UDP handle. The result is a list containing the source IP address, the source port, the destination IP address and the destination port.

All UDP commands are thread safe (allowing you to share data between several programs) like the rest of the CSM TCL commands.

Writing Health Scripts

After you configure each interval of time, an internal CSM scheduler schedules the health scripts. Write the script as if you intend to perform only one probe. You must declare the result of the probe using the **exit** command.

A health script typically performs these actions:

- Opens a socket to an IP address.
- Sends one or more requests.
- Reads the responses.
- Analyzes the responses.
- Closes the socket.
- Exits the script by using exit 5000 (success) or exit 5001 for failure.

This example shows how to probe an HTTP server using a health script:

```
Router(config)# !name = HTTP_TEST

# get the IP address of the real server from a predefined global array csm_env
set ip $csm_env(realIP)
set port 80
set url "GET /index.html HTTP/1.0\n\n"

# Open a socket to the server. This creates a TCP connection to the real server
set sock [socket $ip $port]
fconfigure $sock -buffering none -eofchar {}

# Send the get request as defined
puts -nonewline $sock $url;

# Wait for the response from the server and read that in variable line
set line [ read $sock ]

# Parse the response
if { [ regexp "HTTP/1.. (\[0-9\]+) " $line match status  ] } {
  puts "real $ip server response : $status"
}

# Close the socket. Application must close the socket once the
# is over. This allows other applications and tcl scripts to make
# a good use of socket resource. Health monitoring is allowed to open
# only 200 sockets simultaneously.
close $sock

# decide the exit code to return to control module.
# If the status code is OK then script MUST do exit 5000
# to signal successful completion of a script probe.
# In this example any other status code means failure.
# User must do exit 5001 when a probe has failed.
if { $status == 200 } {
    exit 5000
} else {
    exit 5001
}
```

Exit Codes

The CSM uses exit codes to signify various internal conditions. The exit code information can help you troubleshoot your scripts if they do not operate correctly. You can only use the **exit 5000** and **exit 5001** exit codes.

The TCL script may stop operating for these reasons:

- Syntax errors—Occurs when there is no change in the state of the suspected improper syntax. The syntax error is a stored script control object and can be viewed using the **show mod csm X tech script** command. A suspect will be denied.
- A stopped script—Caused by an infinite loop or caused when the script attempts to connect to an invalid IP address. Each script must complete its task within the configured time interval. If the script does not complete its task, then the script controller terminates the script and the suspect will be failed implicitly.
- Error conditions—Occurs when a connection timeout or a peer refused connection is also treated as an implicit failure.

Table 10-4 shows all exit codes used in the CSM.

Table 10-4 CSM Exit Codes

Exit Code	Meaning and Operational Effect on the Suspect
5000	Suspect is healthy. Controlled by user.
5001	Suspect has failed. Controlled by user.
4000	Script is aborted. The state change is dependent on other system status at that time. Reserved for system use.
4001	Script is terminated. Suspect is failed. Reserved for system use.
4002	Script panicked. Suspect is failed. Reserved for system use.
4003	Script has failed an internal operation or system call. Suspect is failed. Reserved for system use.
unknown	No change.

This example shows how to use the EXIT_MSG variable to track script exit points to detect why a script is not working:

```
set EXIT_MSG "opening socket"
set s [socket 10.2.0.12 80]
set EXIT_MSG "writing to socket"
puts -nonewline $sock $url
```

Use the **show module csm** *slot* **tech script** command to check the EXIT_MSG variable.

Use the errorInfo and errorCode variables that are automatically set by the TCL core to determine why an error occurred. TCL stores internal error information using the errorInfo variable whenever a script task is aborted. Syntax errors, for example, panic, will set these variables with the appropriate information, which you can use to troubleshoot the error.

Environment Variable

Health scripts have access to many configured items through a predefined TCL array. The most common use of this array is to find the current real server IP addresses of the suspect during any particular launch of the script. Table 10-5 lists the members of the csm_env array.

Table 10-5 Member list for the csm_env Array

Member name	Content
realIP	Suspect IP address
realPort	Suspect IP port
intervalTimeout	Configured probe interval in seconds
openTimeout	Configured socket open timeout for this probe
recvTimeout	Configured socket receive timeout for this probe
failedTimeout	Configure failed timeout
retries	Configured retry count
healthStatus	Current suspect health status

You can use the new **probe** *probe-name* **script** command for creating a script probe in Cisco IOS. This command enters a probe submode that is similar to the existing CSM health probe submodes (such as HTTP, TCP, DNS, SMTP, etc.). The probe script submode contains the existing probe submode commands **failed**, **interval**, **open**, **receive**, and **retries**.

A new **script** *script-name* command was added to the probe script submode. This command can process up to five arguments that are passed to the script when it is run as part of the health probe function.

You can also use the corresponding **show module csm** *slot* ***probe script*** **name** *probe-name* **code** command. Refer to the *Catalyst 6500 Series Content Switching Module Command Reference* for release 3.2(1).

System Resource Usage

The Vxworks support application has 255 file descriptors that are divided across all applications, for example, standard input and output, and any socket connections (to or from). When developing standalone scripts, you must be extremely careful when opening a socket. We recommend that you close a socket as soon as the operation is complete because you may run out of resources. The health monitoring module controls the number of open sockets by way of controlling the number of actively running scripts. Standalone scripts do not have this control.

Memory, although a consideration, is not a big limiting factor because the module generally has enough memory available. Each script uses a 128 KB stack, and the rest of the memory is allocated at runtime by the script.

The script tasks are given the lowest priority in the system so that the real-time characteristics of the system remain more or less the same while executing scripts. Unfortunately, scripts having low priority also means that if the system is busy doing non-TCL operations, all TCL threads may take longer to complete. This situation may lead to some health scripts being terminated and the unfinished threads marked failed. To prevent scripts being failed, all script probes should have a retry value of 2 or more. You may want to use native CSM probes (for example, HTTP, DNS, etc.) whenever possible. The scripted health probes should be used to support non-supported applications. The purpose is to provide features, not speed.

TCL supports both synchronous and asynchronous socket commands. Asynchronous socket commands return immediately without waiting for true connections. The internal implementation of the asynchronous script version involves a much more complicated code path with many more system calls per each such command. This condition generally slows down the system by causing some critical

resources to wait while other commands are processing system calls. We do not recommend using the asynchronous socket for scripted probes unless this is a definite requirement. However, you may use this command in a standalone system.

Writing Standalone Scripts

There are no special guidelines for developing a standalone script.

Running TCL Scripts

After a script file has been loaded, the scripts in that file exist in the CSM independent of the file from which that script was loaded. If a script file is subsequently modified, use the **script file** command to reload the script file and enable the changes on the CSM. (Refer to the *Catalyst 6500 Series Content Switching Module Command Reference* for more information.)

The **no script file** command removes the **script file** command from the running configuration. This command does not unload the scripts in that file and does not affect scripts that are currently running on the CSM. You cannot unload scripts that have been loaded. If a loaded script is no longer needed, it is not necessary to remove it; however, do not use it.

In addition to displaying general script information, the **code** version of the **script** command (refer to the *Catalyst 6500 Series Content Switching Module Command Reference*) displays the actual code within the script.

One of the key requirements of porting the TCL standard distribution to the CSM is to modify this base distribution so that multiple vxWorks threads can execute TCL interpreters simultaneously (reentrant behavior). This facility allows many health scripts and other custom scripts to simultaneously execute in multiple TCL interpreters.

Running Probe Scripts

A probe script indicates that the relative health and availability of a real server using the exit code of the script. By calling exit (5000), a script indicates the server successfully responded to the probe. Calling exit (5001) indicates that the server did not respond correctly to the health probe.

Whenever a script probe is executed on the CSM, a special array called csm_env is passed to the script. This array holds important parameters that may be used by the script. Refer to Table 10-5 for a list of the array parameters.

To run a probe script, you must configure a script probe type and then associate a script name with the probe object as follows (refer to the *Catalyst 6500 Series Content Switching Module Command Reference*) as follows:

```
Router(config)# probe probe_name script
Router(config-probe-script)# script script_name [command line arguments]
```

To run a probe script, use this configuration:

```
Router(config)# script file tftp://192.168.10.102/csmScripts
Router(config)# probe ECHOSCRIPT script
Router(config-probe-script)# script echoProbe.tcl
Router(config-probe-script)# interval 10
Router(config-probe-script)# retries 1
Router(config-probe-script)# failed 30
```

Running Standalone TCL Scripts

The **script file** command may be stored in the startup configuration so that it will run when the CSM boots. The script continues to run while the CSM is operating.

To run a TCL script when booting the module, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)#` **`module csm`** *`slot`*	Specifies the module and slot number.
Step 2	`Router(config-module-csm)#` **`script file`** *`file-url`*	Loads the scripts into a script file.
Step 3	`Router(config-module-csm)#` **`show module csm`** *`slot`* **`script task`** [*`index script-index`*] [**`detail`**]	Displays all the loaded scripts.

This example shows how to run a script as a task when booting the CSM:

```
Router(config-module-csm)# configure terminal
Router(config-module-csm)# module csm 4
Router(config-module-csm)# script file tftp://192.168.1.1/httpProbe.test
```

One of the arguments of the **script task** command is *script index*, which is an integer between 1 and 100. The script index is used to halt a specific executing script task instance using the **no script task** command.

To run a standalone TCL script, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)#` **`module csm`** *`slot`*	Specifies the module and slot number.
Step 2	`Router(config-module-csm)#` **`script task`** *`script-index script-name`* [**`arg1`** [**`arg2`**....]]	Runs the script with any command line arguments.
Step 3	`Router(config-module-csm)#` **`show module csm`** *`slot`* **`script`** [**`name`** *`script-name`***`]`** [**`code`**]	Displays the content of all loaded scripts.

This example shows how to run a script as a task when booting the CSM:

```
Router(config-module-csm)# configure terminal
Router(config-module-csm)# Module csm 4
Router(config-module-csm)# script task 1 RECURSIVE_TCL
```

As shown in the examples, use the **show module csm** *slot* **script task index** *script-index* **detail** and the **show module csm** *slot* **script name** *script-name* **code** commands to display information about a specific running script.

Halting TCL Scripts

To stop the script task, enter the **no script task id** command. The task object will be available for troubleshooting and status even after the task finishes executing.

If you need to rerun the same script again, you must do the following:

- Use a different task ID.
- Run the **no script task id** command first, and then retype the command for that script.

Note If you decide to modify the script or the command line arguments while a **script task** command is running, you must stop the task using the **no script task** command and then rerun the task.

Configuring Scripts for Health Monitoring Probes

The CSM supports several specific types of health probes, such as HTTP health probes, TCP health probes, and ICMP health probes when you need to use a diverse set of applications and health probes to administer your network. The basic health probe types supported in the current CSM software release often do not support the specific probing behavior that your network requires. To support a more flexible health-probing functionality, the CSM now allows you to upload and execute TCL scripts on the CSM. (See the "Configuring TCL Scripts" section on page 10-1.)

You can create a script probe that the CSM periodically executes for each real server in any server farm associated with a probe. Depending upon the exit code of such a script, the real server is considered healthy, suspect, or failed. Probe scripts test the health of a real server by creating a network connection to the server, sending data to the server, and checking the response. The flexibility of this TCL scripting environment makes the available probing functions possible.

To load the script file to the switch, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)#` **`module csm`** *`slot`*	Specifies the module and slot number.
Step 2	`Router(config-module-csm)#` **`script file`** *`script file to load`*	Loads the script file to the CSM.
Step 3	`Router(config-module-csm)#` **`show module csm`** *`slot`* **`script`** [**`file`**] [**`name`** *`script-name`*] [**`code`**]]	Displays all loaded scripts.

There are two ways to execute a script on the CSM:

- Creating a script probe
- Performing a standalone script task

As part of a script probe, the script is executed periodically, and the exit code returned by the executing script indicates the relative health and availability of specific real servers. Script probes operate similar to other health probes available in the current implementation of CSM software.

To run the script as a health probe, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)#` **`module csm`** *`slot`*	Specifies the module and slot number.
Step 2	`Router(config-module-csm)#` **`probe`** *`probe-name`* **`script`**	Creates a script probe, and enters the probe script submode.
Step 3	`Router(config-probe-script)#` **`script`** *`script-name`* [**`arg1`** [**`arg`**5]	Sets up to five arguments used by the script when the script executes.
Step 4	`Router(config-module-csm)#` **`show module csm`** *`slot`* **`probe`**	Displays all configured probes.

To run a script as a standalone task, perform this task:

	Command	Purpose
Step 1	`Router(config-module-csm)#` **`module csm`** *`slot`*	Specifies the module and slot number.
Step 2	`Router(config-module-csm)#` **`probe`** *`probe-name`* **`script`**	Creates a script probe, and enters the probe script submode.
Step 3	`Router(config-module-csm)#` **`script task`** *`id script name`*	Runs the script as a stand alone task one time.
Step 4	`Router(config-module-csm)#` **`show module csm`** *`slot`* **`script task`**	Displays all started script tasks.