CHAPTER 10

Using TCL Scripts with the CSM

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

The CSM now enables you to upload and execute Toolkit Command Language (TCL) scripts on the CSM. TCL is a widely used scripting language within the networking community. TCL also has large libraries of developed scripts that can easily be found from various sites. Using TCL scripts, you can 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 the "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.

The CSM currently supports two script modes:

- Probe script mode—These scripts must be written using some simple rules. The execution of these scripts is controlled by health-monitoring module.

  As part of a script probe, the script is executed periodically, and the exit code that is 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.

- Standalone script mode—These scripts are generic TCL scripts. You control the execution of these scripts through the CSM configuration. A probe script can be run as a standalone task.

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.3-3-1.tcl.

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.

Examples for Loading Scripts

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 of 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.

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
script1, file tftp://192.168.1.1/script.master
  size = 40, load time = 03:49:36 UTC 03/26/93
script2, file tftp://192.168.1.1/script.master
  size = 40, 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
```

This example shows how to display the code within a named script:

```
router1# show module csm 6 script name script1 code
script1, file tftp://192.168.1.1/script.master
  size = 40, load time = 03:04:36 UTC 03/06/93
#!name = script1
```

One major difference between a standalone script task and a script probe is that the 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.

Reloading 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.) This example shows how to reload a script:

```
router(config)# module csm 4
router(config-module-csm)# no script file tftp://192.168.1.1/script.master
router(config-module-csm)# script file tftp://192.168.1.1/script.master
Loading script.master from 192.168.1.1 (via Vlan100): !!!!!!!!!!!!!!!
[OK - 74804 bytes]
router(config-module-csm)# end
```

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.

TCL Scripts and the CSM

The CSM release 4.1(1) 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

Table 10-1 TCL Commands Supported by the CSM (continued)

Command			
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		

Table 10-2 lists the TCL command not supported by the CSM.

Table 10-2 TCL Commands Not Supported by the CSM

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

Table 10-3 lists the TCL command specific to the CSM.

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

Table 10-3 CSM Specific TCL Commands

Command	Definition
disable_real *serverfarmName realIp port ,-1* | **all** *probeNumId probeNameId*	Disables a real server from the server farm by placing it in the PROBE_FAIL state. This command returns a 1 if successful and returns a 0 if it fails, as follows: `disable_real SF_TEST 1.1.1.1 -1 10 cisco` **Note** The server farm name must be upper case per the caveat CSCec72471.
enable_real *serverfarmName realIp port ,-1* | **all** *probeNumId probeNameId*	Enables a real server from the PROBE_FAIL state to the operational state. This command returns a 1 if successful and returns a 0 if it fails, as follows: `enable_real SF_TEST 1.1.1.1 -1 10 cisco` **Note** The server farm name must be uppercase per the caveat CSCec72471.
gset *varname value*	Allows you to preserve the state of a probe by setting a variable that is global to all probe threads running from the same script. This command works properly only for probe scripts, not for standalone scripts. Variables in a probe script are only visible within one probe thread. Each time a probe exits, all variables are gone. For example, if a probe script contains a 'gset x 1 ; incr x', variable x would increase by 1 for each probe attempt. • To get the value of variable from script, set *var* or $*var*. • To reset the value of variable from script, unset *var*. • To display the current value of variable, use the **show module csm** *slot* **tech script** command. See the "Debugging Probe Scripts" section on page 10-13 for additional details.

Table 10-3 CSM Specific TCL Commands (continued)

Command	Definition
socket -graceful host A.B.C.D port	By default, all CSM script probes close the TCP socket by sending a reset. This action is taken to avoid the TIME_WAIT state when the CSM initializes an active TCP close. Due to the limitation of 255 sockets available on vxworks, when there are too many probes running at the same time, the CSM can run out of system resources and the next probe attempt will fail when opening the socket. When the socket -graceful command is entered, the CSM closes TCP connections with a FIN instead of a reset. Use this command only when there are fewer than 250 probes on the system, as follows: `set sock [socket -graceful 192.168.1.1 23]`
ping [*numpacket*] *host A.B.C.D*	This command is currently disabled in CSM release 3.2. Allows you to ping a host from a script. This command returns a 1 if successful and returns a 0 if it fails, as follows: `set result [ping 3 1.1.1.1]` **Note** This command blocks the script if the remote host is not in the same subnet as the CSM per caveat CSCea67098.

xml *xmlConfigString*	Sends an XML configuration string to the CSM from a TCL script. This command works only when the XML server is enabled on the CSM. Refer to the XML configuration section.
	This command returns a string with the XML configuration result, as follows:
	`set cfg_result [ xml {`
	`<config>`
	`<csm_module slot="6">`
	`<serverfarm name="SF_TEST">`
	`</serverfarm>`
	`</config>`
	`}`
	`]`

Table 10-4 lists the UDP commands used by the CSM.

Table 10-4 UDP Commands

Command	Definition
udp binary send *handle* [*host port*] *message*	Sends binary data 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 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 close *handle*	Closes the UDP socket associated with handle.
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 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.
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 receive *handle*	Receives a datagram from the UDP socket associated with the handle. This command blocks until a datagram is ready to be received.
udp send *handle* [*host port*] *message*	Sends ASCII data 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.

Probe Scripts

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.

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.

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.

You can use the new **probe** *probe-name* **script** command for creating a script probe in Cisco IOS software. This command enters a probe submode that is similar to the existing CSM health probe submodes (such as HTTP, TCP, DNS, SMTP, and so on). 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.

Example for Writing a Probe Script

This example shows how a script is written 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
}
```

Environment Variables

Health probe 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.

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.

Note The environmental variable information in these sections applies to only probe scripts, not standalone scripts.

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

Exit Codes

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

When a probe script fails and exits with 5001, the corresponding server is marked as PROBE_FAILED and is temporarily disabled from the server farm. The CSM continues to probe the server. When the probe successfully reconnects and exits with 5000, the CSM marks the server status as OPERATIONAL and enables the server from the server farm again.

In addition to script exit 5001, these situations can cause a script to fail and mark the suspect PROBE_FAILED:

- TCL errors—Occur when scripts cointain errors that are caught by the TCL interpreter, for example, a syntax error. The syntax error message is stored in the special variable **erroInfo** and can be viewed using the **show mod csm X tech script** command.
- 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, the script controller terminates the script, and the suspect is failed implicitly.
- Error conditions—Occurs when a connection timeout or a peer-refused connection is also treated as an implicit failure.

Table 10-6 shows all exit codes used in the CSM.

Table 10-6 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.

EXIT_MSG Variable

For debugging purposes, it is a good practice to set the script debug information in a special variable named EXIT_MSG. Using the EXIT_MSG variable, you can track the script execution point by entering specific Cisco IOS **show** commands.

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.

This example shows that the EXIT_MSG was set to "opening socket" because EXIT_MSG is the last command that the script runs before exit:

```
router1# show module csm 4 tech script
SCRIPT CONTROLLER STATS
: ==========================================================================
SCRIPT(0xcbcfb50) stat blk(0xcbcfbb0): TCL_test.tclcbcfb50
CMDLINE ARGUMENT:
 curr_id 1 argc 0 flag 0x0::
 type = PROBE
 task_id = 0x0: run_id = 512 ref count = 2
 task_status = TASK_DONE run status = OK
 start time = THU JAN 01 00:15:47 1970
 end time = THU JAN 01 00:17:02 1970
 runs = 1   +0
 resets = 1   +0
 notrel = 0   +0
 buf read err = 0   +0
 killed = 0   +0
 panicd = 0   +0
last exit status= 4000 last Bad status = 4000
Exit status history:
 Status (SCRIPT_ABORT) occured #(1) last@ THU JAN 01 00:17:02 1970
**TCL Controller:
------------------------
tcl cntrl flag = 0x7fffffff
#select(0) close_n_exit(0) num_sock(1)
 MEM TRACK last alloc(0) last size(0) alloc(0) size(0)
 hm_ver (1) flag(0x0) script buf(0xcbf8c00) new script buf(0x0)  lock owner(0x0) sig
taskdel:0 del:0 syscall:0 syslock:0 sig_select  script ptr (0xcbf88f0) id(0)
Config(0xcbcdd78) probe -> 10.1.0.105:80
 tclGlob(0xcbad050) script resource(0xcbcfa28)
#Selects(0) Close_n_exit(0) #Socket(1)
OPEN SOCKETS:
Last erroInfo = couldn't open socket: host is unreachable
    while executing
"socket 10.99.99.99 80 "
    (file "test.tcl" line 2)
Last errorCode = 65
Last panicInfo =
EXIT_MSG = opening socket
```

Running Probe Scripts

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

The following steps show how to load, create, attach the script to a server farm and virtual server, run the probe scripts, and then display the results:

Step 1 Load the script:

```
router1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
router1(config)# module csm 6
```

```
router1(config-module-csm)# script file tftp://192.168.10.102/csmTcl.tcl
Loading csmTcl.tcl from 192.168.10.102 (via Vlan100): !
[OK - 1933 bytes]
```

Step 2 Create a script probe:

```
router1(config-module-csm)# probe test1 script
rout(config-slb-probe-script)# script CSMTCL
rout(config-slb-probe-script)# interval 10
rout(config-slb-probe-script)# exit
```

Step 3 Attach the probe to the server farm and the virtual server:

```
router1(config-module-csm)# serverfarm test
router1(config-slb-sfarm)# real 10.1.0.105
router1(config-slb-real)# ins
router1(config-slb-real)# probe test1
router1(config-slb-sfarm)# exit
```

Step 4 Attach the server farm to a virtual server:

```
router1(config-module-csm)# vserver test
router1(config-slb-vserver)# virtual 10.12.0.80 tcp 80
router1(config-slb-vserver)# serverfarm test
router1(config-slb-vserver)# ins
router1(config-slb-vserver)# exit
```

At this point, the script probe should be set up. You can use the **show module csm slot tech probe** command to ensure that the scripts are running.

Step 5 Stop the script probe:

```
router1(config-module-csm)# serverfarm test
router1(config-slb-real)# no probe test1
router1(config-slb-sfarm)# exit
```

These examples show how to verify the results of the script commands.

This example shows how to display script information:

```
router1# show module csm 6 script
CSMTCL, file tftp://192.168.10.102/csmTcl.tcl
  size = 1933, load time = 03:09:03 UTC 01/01/70
```

This example shows how to display information about probe scripts:

```
router1# show module csm 6 probe
probe           type    port  interval retries failed  open   receive
------------------------------------------------------------------------
TEST1           script        10       3       300     10     10
router1#
```

This example shows how to display detailed information about a specific probe script:

```
router1# show module csm 6 probe name TEST1 detail
probe           type    port  interval retries failed  open   receive
------------------------------------------------------------------------
TEST1           script        10       3       300     10     10
 Script: CSMTCL
 real                  vserver         serverfarm      policy          status
 ---------------------------------------------------------------------------
 10.1.0.105:80         TEST1           TEST            (default)       OPERABLE
router1#
```

This example shows how to display probe information for real servers:

```
router1# show module csm 6 probe real
  real = 10.1.0.105:80, probe = TEST1, type = script,
    vserver = TEST, sfarm = TEST
    status = FAILED, current = 03:26:04 UTC 01/01/70,
    successes = 1, last success = 03:15:33 UTC 01/01/70,
    failures = 4, last failure = 03:26:04 UTC 01/01/70,
    state = Unrecognized or invalid response
    script CSMTCL
    last exit code = 5001
```

Debugging Probe Scripts

To debug a script probe, you can do the following:

- Use the TCL **puts** command in the scripts running in verbose mode.

  In the verbose mode, a **puts** command causes each probe suspect to print a string to the CSM console. When there are many suspects running on the system, lots of output resources are required or the CSM console might hang. It is very important to make sure that this feature is enabled only when a single suspect is configured on the system.

- Use the special variable EXIT_MSG in the script.

  Each probe suspect contains its own EXIT_MSG variable. This variable allows you to trace the status of a script and check the status of the probe.

  This example shows how to use the EXIT_MSG variable in a script:

  ```
  set EXIT_MSG "before opening socket"
  set s [ socket $ip $port]
  set EXIT_MSG " before receive string"
  gets $s
  set EXIT_MSG "before close socket"
  close $s
  ```

  If a probe suspect fails when receiving the message, you should see EXIT_MSG = before you receive the string.

- Use the **show module csm** *slot* **probe real** [*ip*] command.

  This command shows you the current active probe suspects in the system:

  ```
  router1# show module csm 6 probe real
    real = 10.1.0.105:80, probe = TEST1, type = script,
      vserver = TEST, sfarm = TEST
      status = FAILED, current = 04:06:05 UTC 01/01/70,
      successes = 1, last success = 03:15:33 UTC 01/01/70,
      failures = 12, last failure = 04:06:05 UTC 01/01/70,
      state = Unrecognized or invalid response
      script CSMTCL
      last exit code = 5001
  ```

Note The last exit code displays one of the exit codes listed in Table 10-6 on page 10-10.

- Use the **show module csm** *slot* **tech probe** command.

  This command shows the current probe status (for both the standard and script probe):

```
router1# show module csm 6 tech probe

Software version: 3.2(1)
------------------------------------------------------------
------------------ Health Monitor Statistics ------------
------------------------------------------------------------
 Probe templates: 1
Suspects created: 1
    Open Sockets in System : 8 / 240
    Active Suspect(no ICMP): 0 / 200
    Active Script Suspect  : 0 / 50
    Num events  : 1

Script suspects: 1
  Healthy suspects: 0
Failures suspected: 0
Failures confirmed: 1

Probe attempts:        927  +927
Total recoveries:        3  +3
Total failures:          6  +6
Total Pending:           0  +0
```

- Use the **show module csm** *slot* **tech script** command, and look for the last exit status, persistent variables, errorInfo and EXIT_MSG output:

```
router1# show module csm 6 tech script
SCRIPT(0xc25f7e0) stat blk(0xc25f848): TCL_csmTcl.tclc25f7e0
CMDLINE ARGUMENT:
 curr_id 1 argc 0 flag 0x0::
 type = PROBE
 task_id = 0x0: run_id = 521 ref count = 2
 task_status = TASK_DONE run status = OK
 start time = THU JAN 01 03:51:04 1970
 end time = THU JAN 01 03:51:04 1970

 runs = 13   +11
 resets = 13   +11
 notrel = 0   +0
 buf read err = 1   +1
 killed = 0   +0
 panicd = 0   +0

last exit status= 5001 last Bad status = 5001

Exit status history:

**TCL Controller:
-----------------------
tcl cntrl flag = 0x7fffffff
#select(0) close_n_exit(0) num_sock(2)
 MEM TRACK last alloc(0) last size(0) alloc(0) size(0)
 hm_ver (3) flag(0x0) script buf(0xc25ad80) new script buf(0xc25ad80)
 lock owner(0x0) sig taskdel:0 del:0 syscall:0 syslock:0 sig_select
 script ptr (0xc25f038) id(0)
Config(0xc2583d8) probe -> 10.1.0.105:80
 tclGlob(0xc257010)
SCRIPT RESOURCE(0xc25af70)-------
#Selects(0) Close_n_exit(0) #Socket(2)
OPEN SOCKETS:
```

```
Persistent Variables

--------------------
   x = 11

Last erroInfo =

Last errorCode =
Last panicInfo =
EXIT_MSG = ping failed : invalid command name "ping"
```

The last exit status displays the exit code number (as shown in Table 10-6 on page 10-10).

The Persistent Variables information is set by the **gset varname value** command (as described in Table 10-3 on page 10-5).

The erroInfo variable lists the error that is generated by the TCL compiler. When the script has a TCL runtime error, the TCL interpreter stops running the script and stores the error information in the erroInfo variable.

The EXIT_MSG (see the "EXIT_MSG Variable" section on page 10-10) displays detailed debug information for each probe suspected of failure. Because the output may be lengthy, you can try to filter the keyword first as shown in this example:

```
router1# show module csm slot tech script inc keyword
```

Standalone Scripts

A standalone script is a generic TCL script that loads and runs in the CSM. Because the standalone script is not configured like the probe script is, and it is not attached to a server farm, the script will not be scheduled by the CSM as a periodically run task. To run the task, you must use the **script task** command.

The csm_env environment variables are not applied to a standalone script. You may use the **exit** command, however, if the exit code does not have special meaning for standalone scripts as it does in the probe script.

Example for Writing Standalone Scripts

This example shows how a generic TCL script can be written:

```
#!name = STD_SCRIPT
set gatewayList "1.1.1.1 2.2.2.2"
foreach gw $gatewayList {
    if { ![ ping $gw ]  } {
    puts "-WARNING : gateway $gw is down!!"
  }
}
```

Running Standalone Scripts

A standalone script is a TCL script that will be run once as a single task unlike script probes. The script will run and exit when it is finished. The standalone script will not be run by the CSM periodically unless you configure this script as a task. 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 standalone scripts, perform these steps:

Step 1 Load the script:

```
router1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
router1(config)# module csm 6
router1(config-module-csm)# script file tftp://192.168.10.102/stdcsm.tcl
Loading stdcsm.tcl from 192.168.10.102 (via Vlan100): !
[OK - 183 bytes]
```

Step 2 Run the script as a standalone task:

```
router1(config-module-csm)# script task 1 STD_SCRIPT
```

Step 3 Rerun the script.

You can remove the old task and run it again as follows:

```
router1(config-module-csm)# no script task 1 STD_SCRIPT
router1(config-module-csm)# script task 1 STD_SCRIPT
```

You also can start a new task by giving it a new task ID as follows:

```
router1(config-module-csm)# script task 2 STD_SCRIPT
```

Step 4 Stop the script:

```
router1(config-module-csm)# no script task 1 STD_SCRIPT
```

Step 5 Use the **show** command to display the status of the script:

```
router1#sh mod csm 6 script
STD_SCRIPT, file tftp://192.168.10.102/stdcsm.tcl
router1#sh mod csm 6 script task

task  script                          runs       exit code  status
--------------------------------------------------------------------------
1     STD_SCRIPT                      1          4000       Not Ready
2     STD_SCRIPT                      1          4000       Not Ready
```

To display information about a specific running script, use the **show module csm** *slot* **script task index** *script-index* **detail** or the **show module csm** *slot* **script name** *script-name* **code** commands.

Debugging Standalone Scripts

Debugging a standalone script is similar to debugging a probe script. (See the "Debugging Probe Scripts" section on page 10-13.) You can use the **puts** command in the script to help debugging, because running multiple threads do not cause problems.

TCL Script Frequently Asked Questions (FAQs)

These are some frequently asked questions about TCL scripting for the CSM:

- How are system resources used?

  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 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 that have low priority also mean 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 as 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 or DNS, etc.) whenever possible. The scripted health probes should be used to support nonsupported 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.

- How do I know if a configured probe is running?

  You can run a sniffer on the real server side of the network. Also, you can use the following **show** commands to determine if probes are running on the CSM.

  - If the probe is running, the number of probe attempts should keep increasing as shown in this example:

```
router1# show module csm 6 tech probe
router1#sh mod csm 6 tech probe
Software version: 3.2(1)
------------------------------------------------------------
------------------ Health Monitor Statistics -----------------
------------------------------------------------------------
 Probe templates: 8
Suspects created: 24
    Open Sockets in System : 10 / 240
    Active Suspect(no ICMP): 2 / 200
    Active Script Suspect  : 2 / 50
    Num events  : 24
Script suspects: 24
  Healthy suspects: 16
Failures suspected: 0
Failures confirmed: 8
Probe attempts:        321  +220
```

```
Total recoveries:       16  +0
Total failures:          8  +2
Total Pending:           0  +0
```

– If the probe is running, the success or failures count should increase as shown in this example:

```
router1# show module csm 6 probe real
 real = 10.12.0.108:50113, probe = SCRIPT2_2, type = script,
   vserver = SPB_SCRIPT2, sfarm = SCRIPT2_GOOD, policy = SCRIPT2_GOOD,
   status = OPERABLE, current = 22:52:24 UTC 01/04/70,
   successes = 18, last success = 22:52:24 UTC 01/04/70,
   failures = 0, last failure = 00:00:00 UTC 01/01/70,
   state = Server is healthy.
   script httpProbe2.tcl GET /yahoo.html html 1.0 0
   last exit code = 5000
 real = 10.12.0.107:50113, probe = SCRIPT2_2, type = script,
   vserver = SPB_SCRIPT2, sfarm = SCRIPT2_GOOD, policy = SCRIPT2_GOOD,
   status = OPERABLE, current = 22:52:42 UTC 01/04/70,
   successes = 19, last success = 22:52:42 UTC 01/04/70,
   failures = 0, last failure = 00:00:00 UTC 01/01/70,
   state = Server is healthy.
   script httpProbe2.tcl GET /yahoo.html html 1.0 0
   last exit code = 5000
```

You can also close the socket using FIN in place of reset (RST).

- Why does the UDP probe fail to put the real server in the PROBE_FAIL state when a remote host is unreachable?

  A UDP probe must receive an “icmp port unreachable” message to mark a server as PROBE_FAIL. When a remote host is down or not responding, the UDP probe does not receive the ICMP message and the probe assumes that the packet is lost and the server is healthy.

  Because the UDP probe is a raw UDP probe, the CSM is using a single byte in the payload for probe responses. The CSM does not expect any meaningful response from the UDP application. The CSM uses the ICMP Unreachable message to determine if the UDP application is not reachable.

  If there is no ICMP unreachable reply in the receive timeout, the CSM assumes that the probe is operating correctly. If the IP interface of the real server is down or disconnected, the UDP probe by itself would not know that the UDP application is not reachable. You must configure the ICMP probe in addition to the UDP probe for any given server.

  Workaround: Always configure ICMP with a UDP type of probe.

- Where can I find a script example to download?

  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.3-3-1.tcl.

- Where can I find TCL scripting information?

  The TCL 8.0 command reference is located at this URL:

  http://www.tcl.tk/man/tcl8.0/TclCmd/contents.html

  The TCL UDP command reference is located at this URL:

  http://wwwhome.cs.utwente.nl/~schoenw/scotty/