



TCL'ing Your Cisco Router

Peter J. Welcher

Introduction

This article is being written as the year turns, so I'll wish everyone a Happy New Year, even though you may not see this in print for some weeks yet. This month's article topic is a change of pace and represents a bit of fun to me. I hope you'll enjoy it as much as I have!

The topic is the Tcl programming (scripting) language. Cisco has had it in some routers for a while now, for use in providing Interactive Voice Response (IVR). They've now enhanced it in a way that Service Providers and others may find useful for managing routers. There are some nifty new capabilities in this Cisco IOS embedded Tcl.

We'll be taking a look at those capabilities in this article, complete with some sample Tcl programs. I'm still coming to grips with the idea of being able to run programs in a router, so if you can have a good solid practical use for this, I'd love to hear about it and pass it on to readers. My sample programs below attempt to show some potential uses for the programming capability.

By the way, in case you hadn't noticed, Tcl is pronounced "tickle", making the article title a little punny.

What is Tcl?

Tcl and its associated windowing toolkit Tk are scripting and programming tools that have been around for quite some time now. Tcl and Tk originated with John Ousterhout and others at Berkeley and then Sun starting in the late 80's. I think I first learned about Tcl at a Usenix LISA conference around that time. I've since done some small Tcl programming at various times in the course of the last 10-14 years.

What makes Tcl attractive is that it is small, fairly fast, and often embedded as a secure component of other systems. In particular, it has been used in the performance and test tools community for quite a while. For example, Ixia test systems use Tcl for customized scripting. The Tcl Tk windowing toolkit is portable across Unix and Windows, small but powerful.

For what it's worth, I personally mildly prefer the style of PERL, apparently my brain is wired more for the C, C++, Java and PERL style of coding. Tcl has a bit of a LISP-ish flavor to me, in that you have to think about causing evaluation of commands. This is part of what makes it so powerful, as you can write self-modifying code. You may also have run across Tcl in another setting, as it is the underlying programming language the Expect scripting language is based on. Expect is good at matching CLI prompts and providing responses, turning interactive programs into scripted sessions. There is also a PERL Expect module, which I've found useful in the last year.

This article is not going to attempt to teach you programming in Tcl. I'm hoping you'll get the flavor of it from the examples below. There are some very good resources available at no or low cost for learning Tcl. Here are some links you may find useful. There are a number of good books available (including one by John Ousterhout), I've included the one I have, by Brent B. Welch. I find it quite good. Other books are listed on the www.tcl.tk site.

Purpose	Link
Tcl Developer Exchange	http://www.tcl.tk/
Recommended Book: Brent B. Welch,	http://www.beedub.com/book/

Practical Programming in Tcl and Tk (4th Edition)	
Tcl Tutorial (web version)	http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
Tcl Tutor (recommended!)	http://www.msen.com/~clif/TclTutor.html
Ixia training on Tcl for their test tools	http://www.ixiacom.com/solutions/training/
Ixia Tcl review labs	http://www.ixiacom.com/support/training/TCL202Course.ZIP

Tcl and Cisco Routers

Cisco has embedded Tcl in the Cisco IOS in routers. Not PERL, not Tk. For full details, see the URL http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123newft/123t/123t_2/gt_tcl.htm. This document lists the few changes or restrictions in the Cisco implementation of Tcl.

Pre-requisites:

1. Tcl scripts must be run from tclsh, which is an EXEC mode command requiring enable access.
1. Cisco IOS 12.2(25) S or 12.3(2) T. Cisco IOS 12.3(7) T for the SNMP MIB access via Tcl.

There's an important note in the document at the above URL. **Errors in Tcl scripts can cause infinite loops in the router.** I've experienced this personally, as loss of response in the EXEC session (telnet or console), leading to CPU-HOG syslog messages and eventual router reboot. Conclusion: test your scripts for syntax errors in the lab before trying them on production routers!!!

Testing reveals that the older router code (12.2 vintage) I have, including 12.3 main code train (i.e. non-T), includes Tcl 7.1. The newer code listed above runs Tcl 8.3. This affects the syntax available. The rest of this article will assume you're running 12.3(7) T. Caution: releases 12.3 (7) T through 12.7(11) T seem to have some listed warnings and problems, so it may not be a good idea to run them in production unless you have to.

As far as working with Tcl programs in routers, you'll quickly discover that typing them at the router CLI is not very productive. Instead, I used the 3Com free TFTP/FTP/Syslog server for Windows, available at <ftp://ftp.3com.com/pub/utilbin/win32/3cdv2r10.zip>. You can use any TFTP server of your choice. I set the TFTP server to serve up files in the directory I'm editing Tcl programs in. I can then edit the Tcl program file with Notepad (to avoid file locking issues), save without exiting the editor, and use uparrow on the router to re-issue the Tcl source command (see below). That loads and runs the program. Repeat as needed until you've got the program debugged. That gives you an efficient edit / test / fix cycle. You can easily output messages to yourself for debugging purposes with the Tcl puts command. The comment character is "#", so you can comment out your puts debugging or parts of programs while debugging.

Configuration Options

The Tcl interpreter is invoked by typing "tclsh" at the enable EXEC mode prompt. The Tcl shell attempts to interpret a command as a Tcl command. If that fails, it then tries to interpret the command as a Cisco CLI command. Multiple users can run tclsh at one time -- the interpreter and Tcl server is associated with the tty.

Your Tcl program can issue EXEC or CONFIG mode commands. This allows you to write Tcl programs to build menus or walk junior staff through configuration tasks. It also allows customization of show commands. The Cisco IOS Tcl also allows use of precompiled Tcl, for security and obscurity (including some prevention of program alteration).

As of 12.3(7) T, the Tcl program can also get or set SNMP MIB variables within the router. Testing shows that the SNMP is sourced from 127.0.0.1, and will be blocked if you have an ACL controlling RO or RW SNMP access to the router. On the other hand, you can easily add "permit 127.0.0.1" to such an ACL, and it hardly seems a security concern.

Cisco IOS commands relating to Tcl:

--	--

Command	Purpose
tclsh	As mentioned, launches Tcl configuration mode
scripting tcl encdir tftp://10.1.2.3/enctcl/	Specify default location for external encoding files (advanced topic)
scripting tcl init ftp://user:password@172.17.1.2 /tclscript/initfile1.tcl	Specify initialization script to be run when Tcl shell is started

Cisco extended Tcl with the following commands:

Command	Purpose
ios_config "command" "sub-command" Example: ios_config "interface Ethernet 0" "description this is a demo"	Run a Cisco IOS configuration command (sub-mode commands must be quoted on the same line as the main configuration command)
exec "exec-command" Example: exec "show interfaces"	Run a Cisco IOS EXEC mode command
log_user <i>number</i>	The documentation says "toggles Tcl command output in Tcl configuration mode". That's a bit cryptic, and not quite what I see in the router. Testing shows that log_user 1 causes syntax error messages to show up, log_user 0 hides them.
typeahead <i>string</i>	The documentation says "writes text to the stdin buffer file". This doesn't seem to leave the tclsh interpreter very happy when I try it interactively. I'd like to see a little bit more documentation on this one.
exit	Exit tclsh mode

Samples

Sample 1

```
puts "Hello, world\n"
```

Here's how that works in practice:

```
TheRouter(tcl)#puts "Hello, world\n"
Hello, world
```

```
TheRouter(tcl)#
```

Note that the "\n" caused one newline, and puts also automatically supplied one.

Sample 2

Sample 2 is a variation on the sample `get_bri` function in the [Cisco document above](#), which seemed to beg improvement. I updated the syntax to cause the script to get the names of all the major interfaces, omitting subinterfaces. The regular expression used may need alteration for other router models, i.e. this was not extensively tested. I left some of my debugging puts statements in, commented out, to show how you could work out syntax and other issues.

```

proc get_ints {} {
  #puts "BEFORE\n"
  set check ""
  set int_out [ exec "show interfaces\n" ]

  set mylist [regexp -all -nocase -line -inline {(^[a-z]*[0-9]/*[0-9]*/*[0-9]*)} $int_out]
  #puts "MYLIST $mylist\n"

  foreach int $mylist {
    #puts "INT $int\n"
    if {[string equal $check $int]} {
      if {[info exists ints_out]} {
        append ints_out ", " $int
      } else {
        set ints_out $int
      }
    }
    set check $int
  }

  #puts "AFTER\n"
  return $ints_out
}

puts ""
puts [eval get_ints]

```

Pasting this into my telnet window resulted in:

```

TheRouter(tcl)#proc get_ints {} {
+> #puts "BEFORE\n"
+> set check ""
+> set int_out [ exec "show interfaces\n" ]
+>
+>$-all -nocase -line -inline {(^[a-z]*[0-9]/*[0-9]*/*[0-9]*)} $int_out]
+> #puts "MYLIST $mylist\n"
+>
+>foreach int $mylist {
+> #puts "INT $int\n"
+> if {[string equal $check $int]} {
+> if {[info exists ints_out]} {
+> append ints_out ", " $int
+> } else {
+> set ints_out $int
+> }
+> set check $int
+> }
+> }
+>
+> #puts "AFTER\n"
+> return $ints_out
+>}

TheRouter(tcl)#
TheRouter(tcl)#puts ""

TheRouter(tcl)#puts [eval get_ints]
Ethernet0/0, Serial0/0, Ethernet0/1, Serial0/1, Loopback0, Loopback1

TheRouter(tcl)#

```

Doing this via the Tcl source command looks like the following:

```
TheRouter(tcl)#source tftp://10.20.1.3/pjw02.npd
Loading pjw02.npd from 10.20.1.3 (via Ethernet0/0): !
[OK - 579 bytes]

Ethernet0/0, Serial0/0, Ethernet0/1, Serial0/1, Loopback0, Loopback1

TheRouter(tcl)#
```

I think that starts to suggest a way to use Tcl. If you have built a Tcl program, to use it, you can cut and paste the tclsh and source commands into selected routers. This might for example be useful to filter selected show output before capture. (I've been using CiscoWorks to do the capture, then post-filtering using PERL). One could even use the PERL Expect module to drive connecting and logging into a list of devices, running a Tcl program on each.

Note that maintaining your Tcl programs is going to be a whole lot easier if you keep them on an FTP or TFTP server, rather than copying them to flash on various routers. (Flash memory can also be the source of the sourced program.)

Sample 3

The [Cisco document](#) supplies a sample program showing the Tcl-driven SNMP capability. This is my "improvement" on that basic idea.

```
set ifnumstr [ snmp_getone public 1.3.6.1.2.1.2.1.0]
regexp {val='([0-9]*)'} $ifnumstr {} ifnum
#puts "IFNUM: $ifnum\n"

set vals [ snmp_getbulk public 0 $ifnum \
1.3.6.1.2.1.2.2.1.2 1.3.6.1.4.1.9.2.2.1.1.6 1.3.6.1.4.1.9.2.2.1.1.8]

#Error-check the return value here
if { [regexp -nocase {error} $vals] == 1 } {
    puts "SNMP ERROR: $vals"
    exit 0
}
regsub -all {'} $vals {} val2
regsub -all {\{<obj oid=} $val2 {} val3
regsub -all { val} $val3 {} val4
regsub -all {/>\}} $val4 {} val5
set vals [split $val5]

puts "\nifIndex\tifDescr\t\tifInBps\tifOutBps"

for {set i 0} {$i < 2*$ifnum-1} {incr i 3} {
    regexp {=([A-Za-z/0-9]*)} [lindex $vals $i] {} ifname
    regexp {=([0-9]*)} [lindex $vals [expr 1+$i]] {} ifInBitsSec
    regexp {=([0-9]*)} [lindex $vals [expr 2+$i]] {} ifOutBitsSec
    if {[string length $ifname] < 6} {
        set ifname "$ifname\t"
    }
    puts "[expr $i/2]\t\tifname\t\tifInBitsSec\t\tifOutBitsSec"
}
}
```

Here's a sample run of this on my lab router:

```
TheRouter(tcl)#source tftp://10.20.1.3/pjw03.npd
Loading pjw03.npd from 10.20.1.3 (via Ethernet0/0): !
[OK - 992 bytes]

ifIndex ifDescr          ifInBps ifOutBps
0      Ethernet0/0      1000    2000
1      Serial0/0        0        0
3      Ethernet0/1      0        0
4      Serial0/1        0        0
```

```
6      Null0      0      0
```

```
TheRouter(tcl)#
```

The [Cisco document](#) lists the following Tcl commands for invoking local SNMP.

Command	Purpose
<code>snmp_getbulk community non-rep reps oid1 oid2 ...</code>	Combines get and get-next. Arguments are community string, number of non-repeating OID's, then number of repetitions of get-next for the remaining OID's.
<code>snmp_getid community</code>	Gets basic 6 system SNMP variables.
<code>snmp_getnext community oid1 oid2 ...</code>	Does get-next starting with the specified OID's.
<code>snmp_getone community oid1 oid2 oid3 ...</code>	Does an SNMP get of the OID's listed. (Once, no repetitions).
<code>snmp_setany community oid1 type1 value1 oid2 type2 value2 ...</code>	Does SNMP set of arbitrary number of OID/type/value triples. Type is -i for integer, -u for unsigned32, -c for counter32, -g for gauge, -o for octet string, -d for display string, -ipv4 for IPv4 address, -oid for object ID.

Note that the output from these is a bit ugly. My sample above shows one way to clean the output up. Examples of usage:

```
TheRouter(tcl)#snmp_getid public
{<obj oid='system.1.0' val='Cisco IOS Software, C2600 Software (C2600-IX-M), Ver
sion 12.3(11)T, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2004 by Cisco Systems, Inc.
Compiled Sat 18-Sep-04 11:38 by eaarmas' />}
{<obj oid='system.2.0' val='products.186' />}
{<obj oid='sysUpTime.0' val='145043' />}
{<obj oid='system.4.0' val='Kilroy J. Smith' />}
{<obj oid='system.5.0' val='TheRouter.cisco.com' />}
{<obj oid='system.6.0' val='The Basement' />}
```

```
TheRouter(tcl)#snmp_getone public 1.3.6.1.2.1.1.4.0
{<obj oid='system.4.0' val='Kilroy J. Smith' />}
```

```
TheRouter(tcl)#snmp_setany public 1.3.6.1.2.1.1.4.0 -d "John Doe"
{<snmp error type='tcl_snmp_processing_error' value='6' text='NO_ACCESS_ERROR: 1
.' />}
TheRouter(tcl)#snmp_setany private 1.3.6.1.2.1.1.4.0 -d "John Doe"
{<obj oid='system.4.0' val='John Doe' />}
```

```
TheRouter(tcl)#snmp_getid public
{<obj oid='system.1.0' val='Cisco IOS Software, C2600 Software (C2600-IX-M), Ver
sion 12.3(11)T, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2004 by Cisco Systems, Inc.
Compiled Sat 18-Sep-04 11:38 by eaarmas' />}
{<obj oid='system.2.0' val='products.186' />}
{<obj oid='sysUpTime.0' val='159727' />}
{<obj oid='system.4.0' val='John Doe' />}
{<obj oid='system.5.0' val='TheRouter.cisco.com' />}
{<obj oid='system.6.0' val='The Basement' />}
```

```
TheRouter(tcl) #
```

I included the last bit above to show that indeed the sysContact info had changed.

Tcl for Other Purposes

The [Cisco document](#) shows Tcl procedures that you can use to have your router to send you SMTP email. Note that Tcl gives you the ability to open sockets. (Hmm, generate TCP or UDP traffic, e.g. for load testing?)

IVR (Interactive Voice Response). See <http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/vcl.htm> and especially <http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t11/ivrapp/index.htm>. There's definitely some major reading there!

ESM (Embedded Syslog Manager) -- see also http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123newft/123t/123t_2/gt_esm.htm. We're out of time for now. Concerning what it does: "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."

Summary

I hope you enjoyed this as much as I did. (If not, you probably didn't make it to this point anyway.) I realize most networking folks aren't going to run out and start doing Tcl programming. But it's different enough that you might think it's fun the way I do! And this gives us the chance to see the sorts of hooks that Cisco has now built in, to allow Service Provider OSS's, Enterprises, and network management tools to do more powerful things with Cisco routers in the future.

Your comments, questions, and suggestions for future articles are of course welcome! See below to decipher my email address.

Dr. Peter J. Welcher (CCIE #1773, CCSI #94014, CCIP) is a Senior Consultant with Chesapeake NetCraftsmen. NetCraftsmen is a high-end consulting firm and Cisco Premier Partner dedicated to quality consulting and knowledge transfer. NetCraftsmen has ten CCIE's, with expertise including large network high-availability routing/switching and design, VoIP, QoS, MPLS, IPsec VPN, wireless LAN and bridging, network management, security, IP multicast, and other areas. See <http://www.netcraftsmen.net> for more information about NetCraftsmen. Pete's links start at <http://www.netcraftsmen.net/welcher>. New articles will be posted under the Articles link. Questions, suggestions for articles, etc. can be sent to [pjw <at> netcraftsmen <dot> net](mailto:pjw@netcraftsmen.net) (formatted this way to fool email harvesting software).

1/3/2005

Copyright (C) 2005 Peter J. Welcher