

Automating Echola's Layer1 Optical Switches

In order to automate Echola's Layer 1 switches you would need to write **tcl/expect** based script. You can also use perl/xml. See an example of perl script at the end of the document. The tcl and expect scripting languages are easy to learn. We have given an example script written for fc811/fc1611 at the end which you can modify to suit your need. There are tons of online sources for learning tcl & expect. The following provides quick high level overview of tcl and expect <http://cplug.org/t/uploads/2009/02/tcl-expect.pdf>. There is a good book from O'Reilly which provides great insight into expect language itself: "Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs (Nutshell Handbooks)".

Running scripts from Unix/Linux systems

If you want to run the script from a Unix/Linux based machines then there is possibility that you may be already having these tools on your system. Check if it's already been installed by typing "expect" from Unix/Linux prompt. If it is not then you will have to install it using package install tool for that particular flavor of Unix/Linux. For instance, on Fedore core Linux, you can use "yum install tcl expect" to install tcl and expect.

Running scripts from Windows

For windows based systems you can install windows free community version of ActiveTcl from Activestate <http://www.activestate.com/activetcl/downloads>. The expect is not available yet for 64bit version of Windows 7/Vista. So you will need to download 32bit version for ActiveTcl first and then make sure to install "expect" using command "teacup install Expect".

Also you need to enable "telnet" client on Windows before running any scripts. In order to enable telnet on Windows follow these steps

- Start
- Control Panel
- Programs And Features
- Turn Windows features on or off
- Check Telnet Client
- Hit OK

After that you can start Telnet via Command Prompt to check if it works.

The following sample script actually login into fc1611/fc811 switch and issue a switch command then check whether the switch command was successful and return the result before terminating the telnet session. This script takes argument (port number and state of the port (on/off)) from commands line argument. Cut and Paste the following script on to any editor and save as "rosctl". Then you can run the script by issuing **rosctl -p <port#> on|off**. For instance, to switch port 2 to ON, you can call script as **rosctl -p 2 on**. Make sure you have proper path set for expect on first line "#!/usr/bin/expect" for Unix/Linux based systems. For windows you will have to uncomment 'exec' and 'package' commands as mentioned in the script. All comments inside '#' provide more info on what the script is doing.

The Sample Tcl/Expect script

```
#!/usr/bin/expect
#####
# This script switches the given port and verifies if the port is switched from a remote machine
# Command Usage: rosctl -p <port#> on|off
#####
# For windows uncomment following
# exec tclsh "$0" ${1+"$@"}
# package require Expect

# Check number of arguments passed to this command if < 3 then spit out error & exit

if { $argc < 3 } {
    puts "Usage: rosctl -p <port#> on|off\n"
    exit 1
}

# Set telnet host, username, password and other parameters, modify these to reflect your setup

set hostname "192.168.2.20"
set username "osctl"
set password "osctl"
set prompt "osctl@.*\$"
set port [lindex $argv 1]
set status [lindex $argv 2]
set commandcontrol "osctl -p $port $status"
set commandstatus "osctl -s $port"

# Display info.

puts "Connecting to $hostname."

# Connect to the telnet server using the "spawn" command.

spawn telnet $hostname
#spawn C:\Putty\putty.exe -telnet $hostname

# Wait for a login prompt.

expect -re "(Name|login|Login|Username).*:.*" {

    # Login prompt received. Send user name to fc811/fc1611.

    send "$username\r"
```

```

} eof {

    # No login prompt received. Display an error.

    puts "could not connect\n"
}

# Wait for a password prompt from the Unix server.

expect "Password:" {

    # Password prompt received. Send the password.

    send "$password\r"
}

# Wait for the switch prompt.

expect -re $prompt {

    # Issue osctl command to switch given port

    send "$commandcontrol\r"
}

# Wait for the switch prompt again to check status.

expect -re $prompt {

    # Issue osctl command to check status

    send "$commandstatus\r"
}

# Discard echoed command - we need only the status

expect "$commandstatus\r"

# Discard unwanted prompt as well

expect -re "(.*)$prompt"

#Debug
#puts "\nGOT*****$expect_out(buffer)*****\n"
#puts "\n GOTS #####$expect_out(1,string)#####\n"

# Save remaining to buffer 'data'

```

```
set data $expect_out(1,string)
```

```
# Check return status and display result accordingly
```

```
switch -re $data {  
  "off" { puts "Port $port is OFF" }  
  "on" { puts "Port $port is ON" }  
  default { puts "Port $port status is unknown" }  
}
```

```
# Terminate telnet
```

```
send "exit\r"
```

The Sample Perl/xml script

```
#####  
# Command Syntax: osctlauto.pl http://<ipaddress> <port#> <on|off> #  
#####  
use LWP::UserAgent;  
use XML::Simple;  
  
$IPADDR = shift;  
$PORT = shift;  
$STAT = shift;  
  
# create objects  
$xml = new XML::Simple;  
$ua = LWP::UserAgent->new;  
  
# set status on/off  
  
$REQUEST=$IPADDR . "/xmlapi/setstatus.php?port=" . $PORT . "&" . "status=" . $STAT;  
#print "REQUEST: $REQUEST\n";  
$req = HTTP::Request->new(GET => $REQUEST);  
$req->header('Cookie' => 'test=quest');  
$res = $ua->request($req);  
  
# check status  
  
$REQUEST=$IPADDR . "/xmlapi/getstatus.php?port=" . $PORT;  
print "REQUEST: $REQUEST\n";  
$req = HTTP::Request->new(GET => $REQUEST);  
$req->header('Cookie' => 'test=quest');  
$res = $ua->request($req);  
$data = $xml->XMLin($res->content);  
  
#print status  
  
print "STATUS = $data->{status}\n";
```