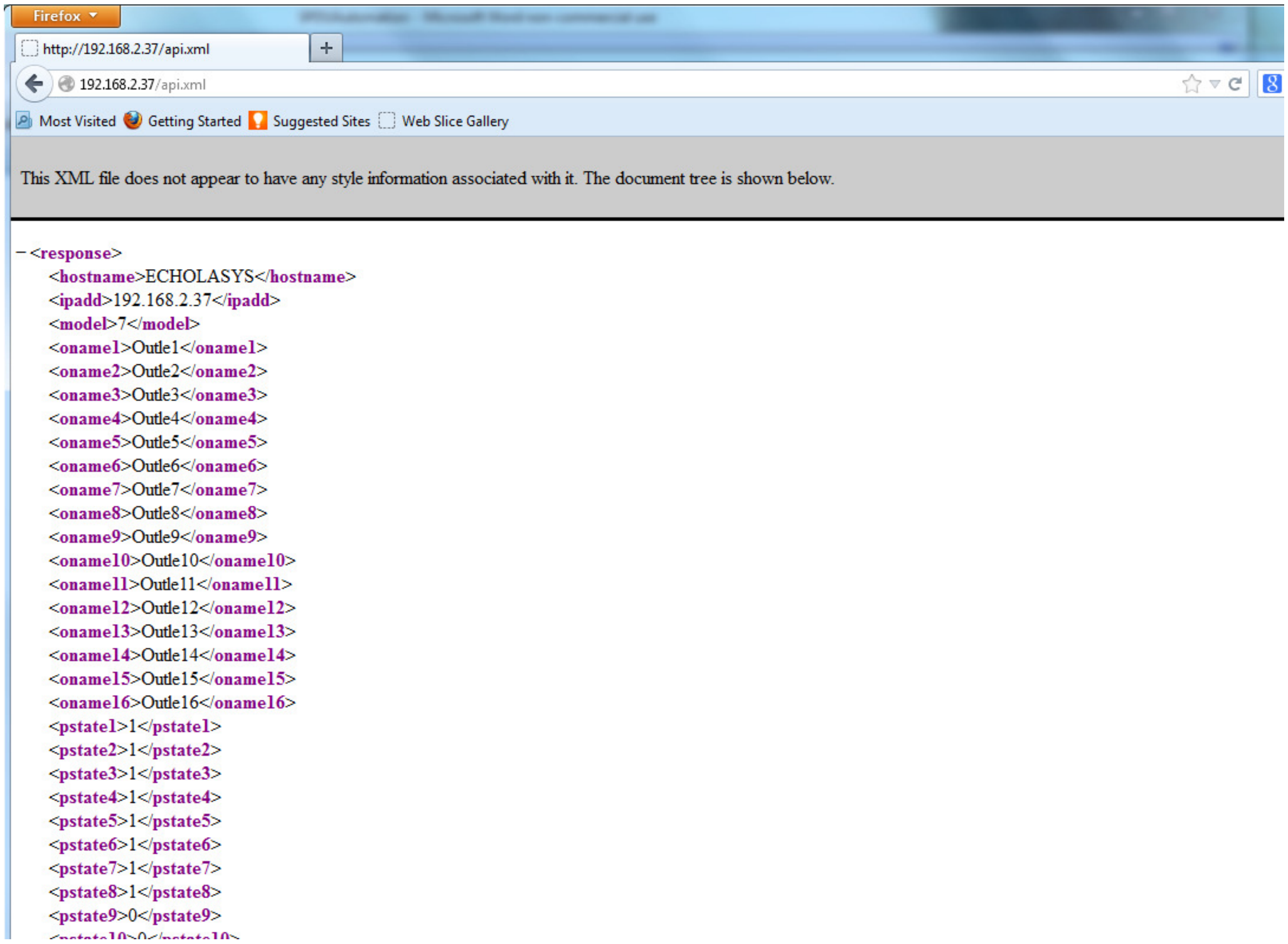
Automating Echola's smart PDUs

To write a script to automate Echola's smart PDUs you can use **perl/xml/http**, **tcl/expect/telnet or SNMP**.

Using Perl

Most of the PDU instantaneous information is available through file called api.xml.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <response>
    <hostname>ECHOLASYS</hostname>
    <ipadd>192.168.2.37</ipadd>
    <model>7</model>
    <oname1>Outle1</oname1>
    <oname2>Outle2</oname2>
    <oname3>Outle3</oname3>
    <oname4>Outle4</oname4>
    <oname5>Outle5</oname5>
    <oname6>Outle6</oname6>
    <oname7>Outle7</oname7>
    <oname8>Outle8</oname8>
    <oname9>Outle9</oname9>
    <oname10>Outle10</oname10>
    <oname11>Outle11</oname11>
    <oname12>Outle12</oname12>
    <oname13>Outle13</oname13>
    <oname14>Outle14</oname14>
    <oname15>Outle15</oname15>
    <oname16>Outle16</oname16>
    <pstate1>1</pstate1>
    <pstate2>1</pstate2>
    <pstate3>1</pstate3>
    <pstate4>1</pstate4>
    <pstate5>1</pstate5>
    <pstate6>1</pstate6>
    <pstate7>1</pstate7>
    <pstate8>1</pstate8>
    <pstate9>0</pstate9>
```

You can use any xml parser to parse the output of api.xml. As you can see from above output, hostname can be parsed using tag 'hostname'; similarly ip address is parsed using the tag 'ipadd'. The tag 'oname1' is for outlet1's name and 'oname2' is for outlet2's name and so on. The pstat1-16 shows the status of outlet 1-16. Here 1 means 'on' and 0 means 'off'. The pow1-16 shows the power consumption of outlet1-16. The 'temp' and 'volt' shows temperature and line voltage/frequency respectively. The temperature is shown in Celsius but you need to add 22°C (base) to this value to get actual value. In order to convert this value to Fahrenheit, use formula (°C+22)*9/5 + 32. The 'reportd' shows cumulative energy for each “hour” up to 24 hours (last 24 hours). The ‘reportm’ shows cumulative energy for each “day” up to 31 days (last 31 days).

Here is an example how to use Perl script which uses api.xml file from the PDU to parse some of its output. If you are on Windows you can use strawberry perl to run this script.

```
###############################################################
#     Command Syntax: perl xmlget.pl http://<ipaddress>                      #
###############################################################
use LWP::UserAgent;
use XML::Simple;
$IPADDR = shift;

# create objects
$xml = new XML::Simple;
$ua = LWP::UserAgent->new;

# send request for api.xml
$REQUEST=$IPADDR . "/api.xml";
$req = HTTP::Request->new(GET => $REQUEST);
$req->header('Cookie' => 'test=quest');
$res = $ua->request($req);
$data = $xml->XMLin($res->content);

# print them
print  "Outlet1 Power: $data->{pow1}";
print  "Outlet2 Power: $data->{pow2}";
print  "Outlet3 Power: $data->{pow3}";
print  "Outlet4 Power: $data->{pow4}";
print  "Outlet5 Power: $data->{pow5}";
print  "Outlet6 Power: $data->{pow6}";
print  "Outlet7 Power: $data->{pow7}";
print  "Outlet8 Power: $data->{pow8}";
print  "Voltage/Frequency: $data->{volt}”;
print  "Temperature: $data->{temp}”;
```

Using TCL

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 smart PDU 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 "switch". Then you can run the script by issuing **switch on|off|reset <outlet#> .** For instance, to switch outlet 2 to ON, you can call script as **switch on 2**. 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: switch on|off |reset  <outlet#>
################################################################################
# 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: switch on|off  <outlet#>\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 "admin"
    set password "admin"
    set prompt "ECHOLASYS@.*\$"
    set state [lindex $argv 1]
    set outlet [lindex $argv 2]
    set commandcontrol "switch  $state $outlet "
    set commandstatus "show status  $outlet"

# Display info.

    puts "Connecting to $hostname."

# Connect to the telent 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 smart PDU.

       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 $outlet is OFF" }
               "on"  { puts "Port $outlet is ON" }
              default { puts "Port $outlet status is unknown" }
      }

# Terminate telnet

      send "exit\r"
```