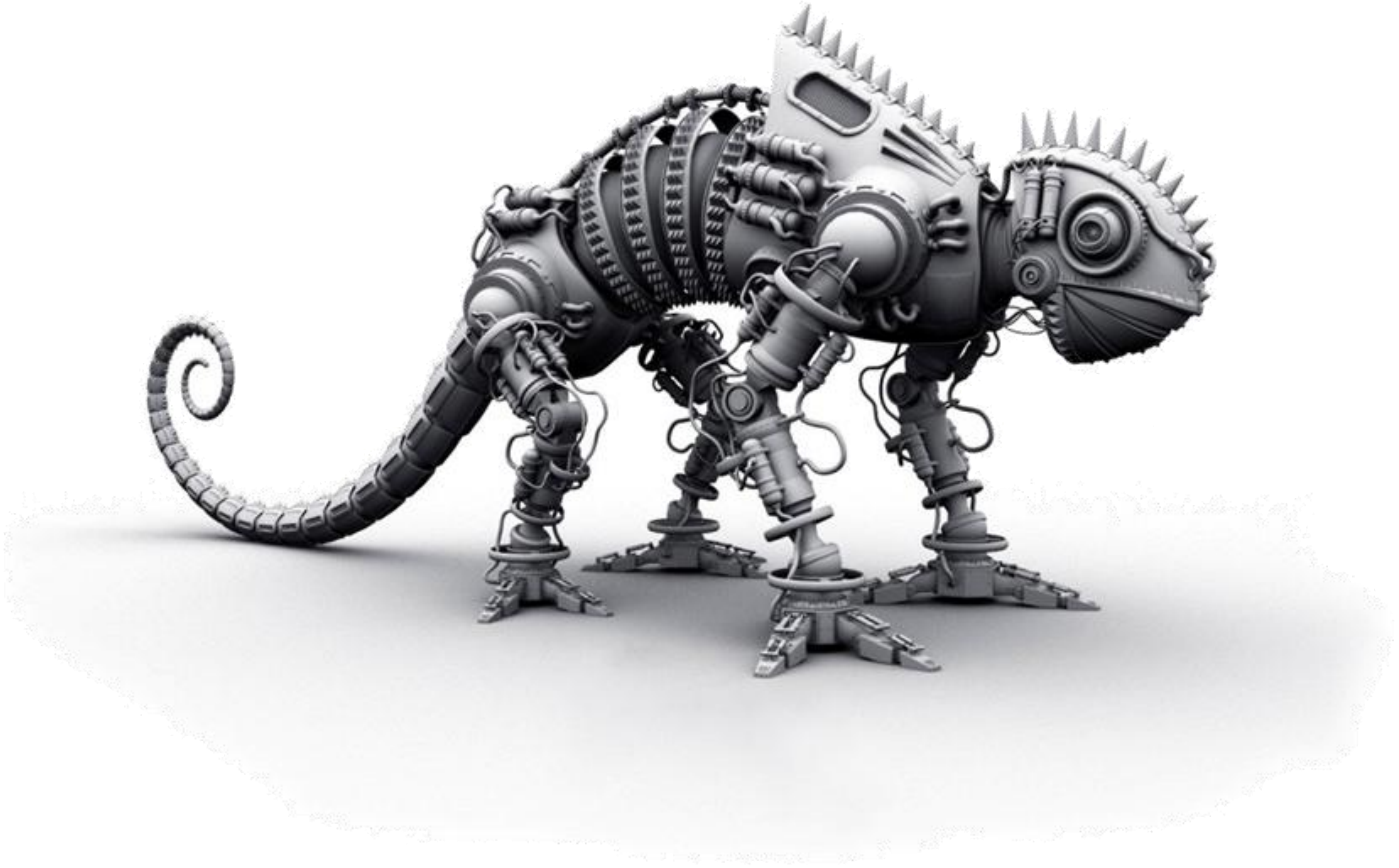
XenaScripting

Test Automation – Step by Step



AGENDA

What is XenaScripting?

Command Line Interface (CLI) to the Xena Server (chassis)

- Covers all functionality.

Supports interactive sessions

- Command and Reply
- Replay of command lists

Scripted or programmed sessions

- Pre-programmed actions and check

Reuse of existing GUI setup

- Jump start script writing
- Use as template

XenaScripting Advantages

Versatility

- Can be used with ANY scripting and program language
- Can run in ANY operating system
- No need to install drivers or proprietary programs
- Re-use your existing automation framework

Fast

- Extremely low communication overhead
- Very simple and efficient protocol
- No requirements to the CLI client

Simplicity

- Purely text based
- No binary modules

Fast Learning curve

- Very easy to learn
- Fully documented
- Very easy to debug
- Reuse existing configurations

CLI API - BASICS

Xena Automation is based on:

1. A TCP connection from script/program to Xena tester

+

2. Input/output via CLI-like API commands over the TCP socket

TCL Example

Creating a TCP Socket –

```
set s [socket -async $chassis_ip $chassis_port]
```

Communicating via Socket (s) –

```
puts $s "c_logon $chassis_pass"
gets $s response
```

CLI API - BASICS

Xena Automation is based on:

1. A TCP connection from script/program to Xena tester

+

2. Input/output via CLI-like API commands over the TCP socket

Ruby Example:

Creating a TCP Socket –

```
require 'socket'      # Sockets are in standard library
s = TCPSocket.open($hostname, $port)
```

Communicating via Socket (s) –

```
socket.puts(tx_string)
response = socket.gets
```

CLI API - BASICS

Xena Automation is based on:

1. A TCP connection from script/program to Xena tester

+

2. Input/output via CLI-like API commands over the TCP socket

Python Example:

Creating a TCP Socket –

```
from LabUtils.Drivers.SocketDrivers import SimpleSocket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Communicating via Socket (s) –

```
sock.send(cmd + '\n')
response = sock.recv(1024)
```

CLI API - BASICS

Xena Automation is based on:

1. A TCP connection from script/program to Xena tester

+

2. Input/output via CLI-like API commands over the TCP socket

Bash Example:

Creating a TCP Socket –

```
exec 3<> /dev/tcp/${MACHINE}/${PORT}
```

Communicating via Socket (s) –

```
echo -en "C_LOGON ${PASSWORD}\r\n" >&3
read  <&3
```

WORTH REMEMBERING

TCP Port used for connecting:	2261
Once TCP Socket is open, ASCII text commands are used, must be terminated by:	CR/LF
To keep session alive and avoid timeout, use:	C_TIMEOUT 99999

CLI API - Commands

Command Structure

```
module/port command [index] value value …
```

Example:

```
2/5 PS_RATEPPS [3] 500000
```

Meaning:

- Module is no. 2
- Port on module is no. 5
- Command is: **`PS_RATEPPS`** =Set Stream Rate as Packets Per Second
- Stream on port is no. 3 *(SID)*
- Packets Per Second is set to 500000

- <u>All indices start at zero</u>

* Use “**?**” as value to READ current setting (explained in next slide)

CLI API - Commands

Command Structure

Most commands can be used both to :

query the current status/data using "?" or

modify/set data via the actual value you would like to set:

You would query for the current value this way:

```
0/5 PS_RATEPPS [3] ?
```

And the chassis would respond the same way that you set the value yourself:

```
0/5 PS_RATEPPS [3] 500000
```

CLI API - Commands

Command Identifiers

C_........CHASSIS PARAMETER
M_...... MODULE PARAMETER
P_...... PORT PARAMETER
PS_.....STREAM PARAMETER
PM_.... MATCH TERM PARAMETER
PL_..... LENGTH TERM PARAMETER
PF_......FILTER PARMATERS
PC_.... CAPTURE PARAMETER
PT_.... TRANSMIT STATISTICS PARAMETER
PR_.... RECEIVE STATISTICS PARAMETER
PD_.... DATASET PARAMETER(HISTOGRAMS)
PP_.... 40/100G PARAMETER

CLI API - Commands

Special scripting commands

Commands for supporting the scripting process itself:

sync	Produces a reply of <SYNC>, helpful when parsing and delimiting returned lines
sync on	Automatically set “sync” after each command.
sync off	Disables “sync on”.
wait n	Waits for the specified number of seconds, (max 60), then replies <RESUME>
help ?	Gives an overview of the built-in help function
help ”cmd”	Gives a brief overview of the syntax for “cmd”

CLI API – Status Messages

Status messages

The set/change commands themselves simply produce a reply from the chassis of:

```
<OK>
```

If something is unacceptable to the chassis, it may return one of the following:

<NOTLOGGEDON>	You have not issued a C_LOGON providing the chassis password.
<NOTRESERVED>	You have not issued a x_RESERVATION for the resource you want to change.
<NOTWRITABLE>	The parameter is read-only.
<NOTREADABLE>	The parameter is write-only.
<NOTVALID>	The operation is not valid in the current chassis state, e.g. because traffic is on.
<BADMODULE>	The module index value is out of bounds.
<BADPORT>	The port index value is out of bounds.
<BADINDEX>	A parameter sub-index value is wrong.
<BADSIZE>	The size of a data value is not appropriate.
<BADVALUE>	A value is not appropriate.
<FAILED>	An operation failed to produce a result.

CLI API – Status Messages

Status messages

If there is a plain syntax error, misspelled parameter, or an inappropriate use of module/port/indices, the chassis will return a line pointing out the column where the error was detected, e.g.:

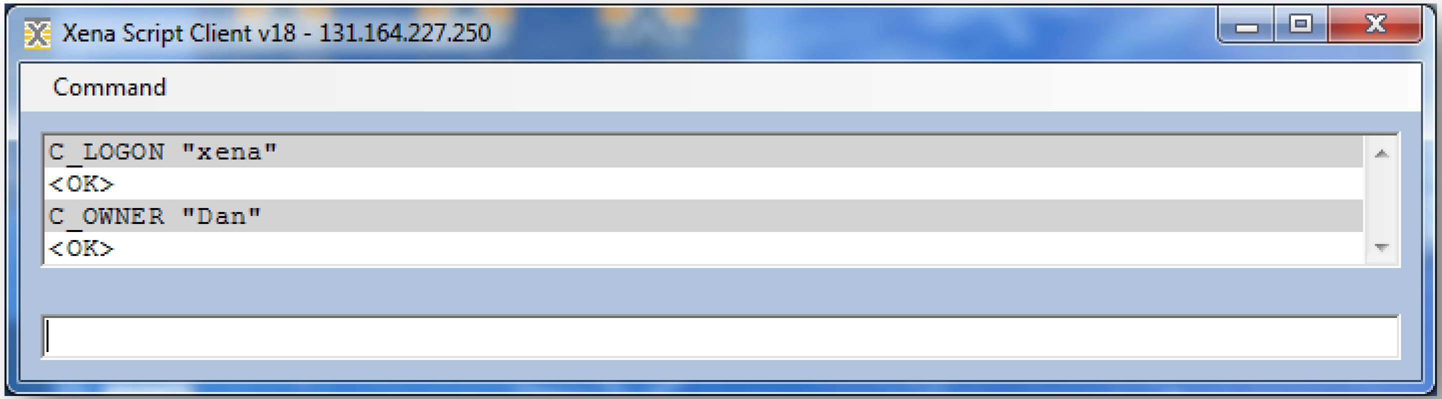
```
0/5 PS_RATEPPS [] 5q00
                 ---^
#Syntax error in column 24
```

CLI API – First Steps

First open connection via script/program or via Xena Script Client.

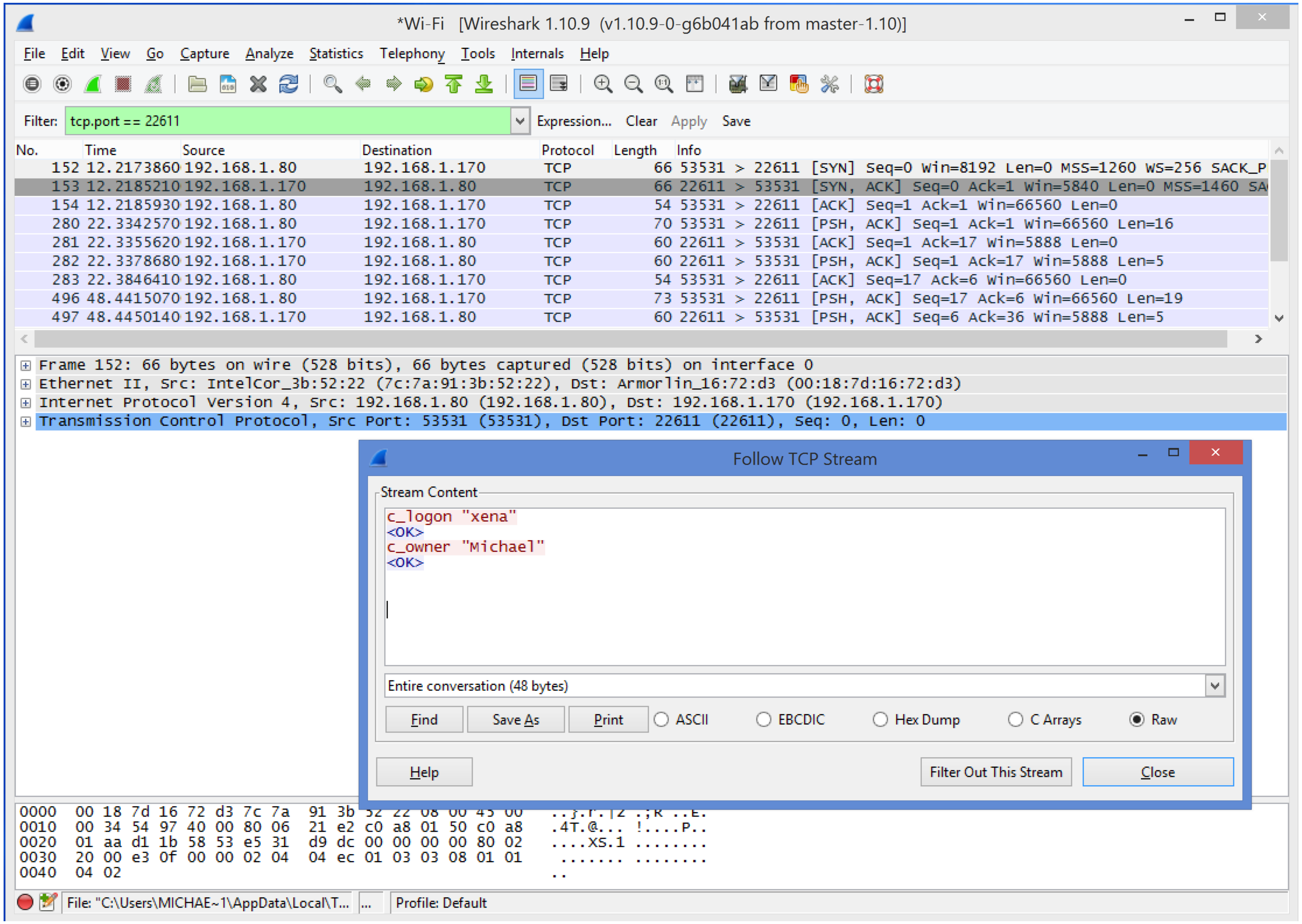
Than we authenticate the connection to the chassis and provide a user name for reservation:

```
C_LOGON "xena"
C_OWNER "example"
```



```
C_LOGON "xena"
<OK>
C_OWNER "Dan"
<OK>
```

CLI API - First Steps



*Wi-Fi [Wireshark 1.10.9 (v1.10.9-0-g6b041ab from master-1.10)]

Filter: tcp.port == 22611

```
No.  Time        Source         Destination    Protocol Length Info
152  12.2173860  192.168.1.80   192.168.1.170  TCP      66     53531 > 22611 [SYN] Seq=0 Win=8192 Len=0 MSS=1260 WS=256 SACK_P
153  12.2185210  192.168.1.170  192.168.1.80   TCP      66     22611 > 53531 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SA
154  12.2185930  192.168.1.80   192.168.1.170  TCP      54     53531 > 22611 [ACK] Seq=1 Ack=1 Win=66560 Len=0
280  22.3342570  192.168.1.80   192.168.1.170  TCP      70     53531 > 22611 [PSH, ACK] Seq=1 Ack=1 Win=66560 Len=16
281  22.3355620  192.168.1.170  192.168.1.80   TCP      60     22611 > 53531 [ACK] Seq=1 Ack=17 Win=5888 Len=0
282  22.3378680  192.168.1.170  192.168.1.80   TCP      60     22611 > 53531 [PSH, ACK] Seq=1 Ack=17 Win=5888 Len=5
283  22.3846410  192.168.1.80   192.168.1.170  TCP      54     53531 > 22611 [ACK] Seq=17 Ack=6 Win=66560 Len=0
496  48.4415070  192.168.1.80   192.168.1.170  TCP      73     53531 > 22611 [PSH, ACK] Seq=17 Ack=6 Win=66560 Len=19
497  48.4450140  192.168.1.170  192.168.1.80   TCP      60     22611 > 53531 [PSH, ACK] Seq=6 Ack=36 Win=5888 Len=5
```

```
Frame 152: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: IntelCor_3b:52:22 (7c:7a:91:3b:52:22), Dst: Armorlin_16:72:d3 (00:18:7d:16:72:d3)
Internet Protocol Version 4, Src: 192.168.1.80 (192.168.1.80), Dst: 192.168.1.170 (192.168.1.170)
Transmission Control Protocol, Src Port: 53531 (53531), Dst Port: 22611 (22611), Seq: 0, Len: 0
```

Follow TCP Stream

Stream Content

```
c_logon "xena"
<OK>
c_owner "Michael"
<OK>
```

Entire conversation (48 bytes)

( ) ASCII ( ) EBCDIC ( ) Hex Dump ( ) C Arrays [✓ Raw]

```
0000  00 18 7d 16 72 d3 7c 7a  91 3b 52 22 08 00 45 00   ..}.r.|z .;R"..E.
0010  00 34 54 97 40 00 80 06  21 e2 c0 a8 01 50 c0 a8   .4T.@... !....P..
0020  01 aa d1 1b 58 53 e5 31  d9 dc 00 00 00 00 80 02   ....XS.1 ........
0030  20 00 e3 0f 00 00 02 04  04 ec 01 03 03 08 01 01    ....... ........
0040  04 02                                              ..
```

File: "C:\Users\MICHAE~1\AppData\Local\T... ... Profile: Default

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save
Find
Save As
Print
Help
Filter Out This Stream
Close

Scripting...

An automated test script has two main components:

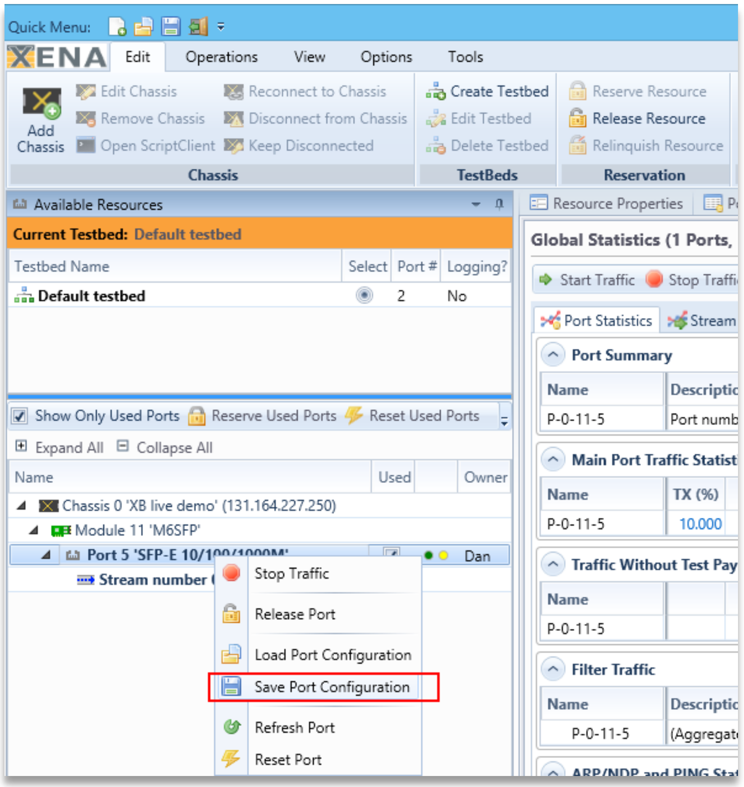
1 - Port Configuration

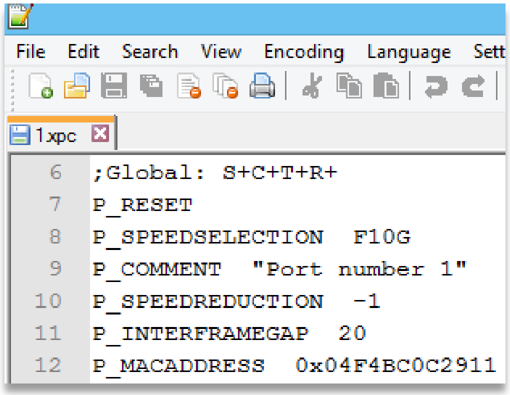
2 - Test Logic

Scripting... – Port Configuration

Port Configuration

Includes *Port, Streams, Capture, Filters, Histograms* Configuration commands
Can be exported via the GUI
Can be imported to a port via a small method

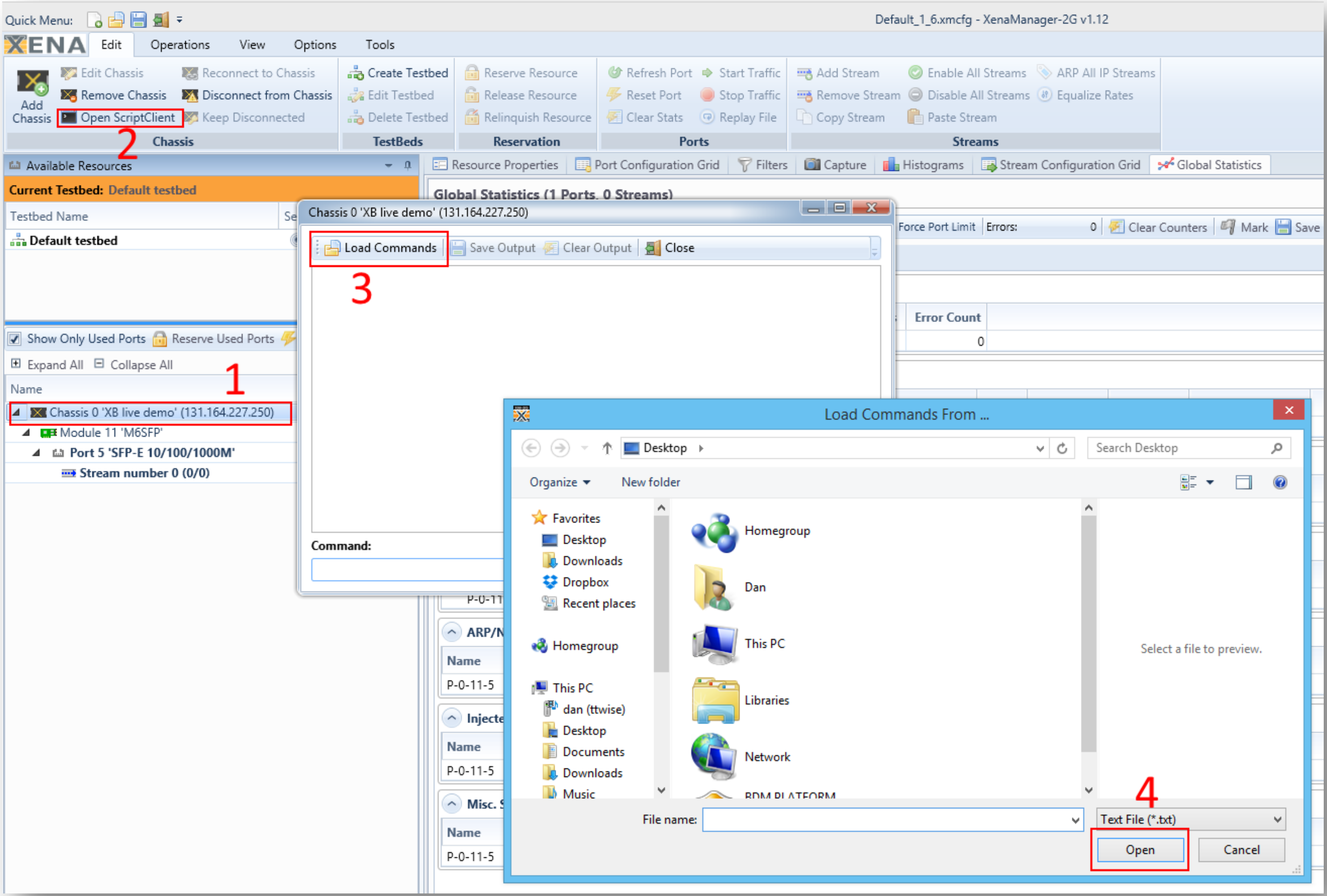




```
;Global: S+C+T+R+
P_RESET
P_SPEEDSELECTION  F10G
P_COMMENT  "Port number 1"
P_SPEEDREDUCTION  -1
P_INTERFRAMEGAP  20
P_MACADDRESS  0x04F4BC0C2911
```

```
proc LoadPortConfig { s port file_name console} {
    set file_data [read $fp]
    set data [split $file_data "\n"]

    foreach line $data {
        if {$line==""} {break}
        if !{[string match ";*" $line]} {
            puts $s "$port $line"
            gets $s response
        }
    }
    close $fp
}
```

Scripting... – Port Configuration (reverse)



Scripting… – Test Logic

Test Logic

If/else/while/etc… logic provided by the scripting or program language

Real-time interaction commands with Xena referred to as the SCRIPT CORE COMMANDS

Real-time interaction commands with the DUT

Connect

- Connect / Login

Reservation

- Reserve / Release / +IsReserved? / + IsReservedByMe?

Port

- PortLinkUp / PortLinkDown / +HasLink?

Configuration

- Clear / Load

Traffic

- Start / Stop

Capture

- Start / Stop / Save

Results

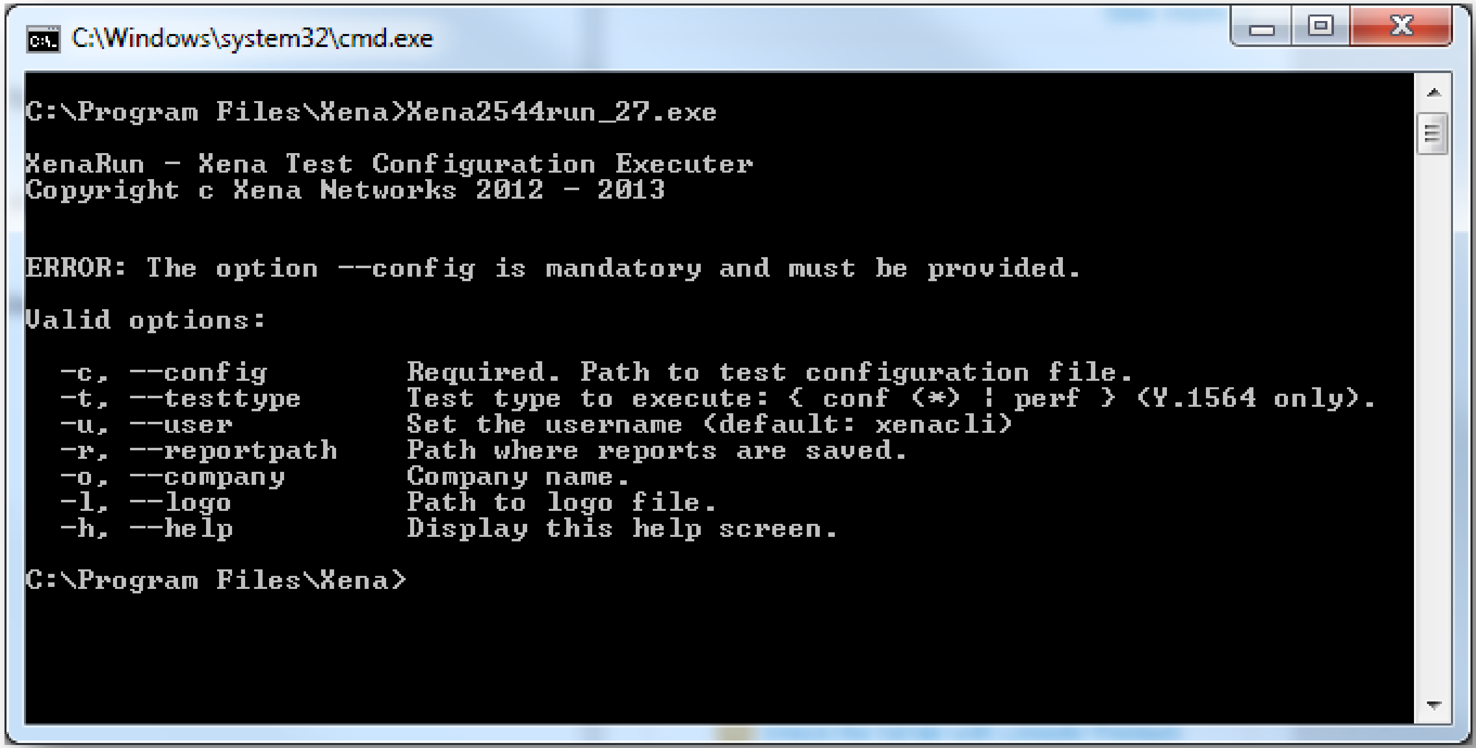
- Clear / Get (Port / Stream / Filter) / Save (Port / Stream / *Filter*) to .CSV

Scripting... – Login Example

```
# ---------------- Login ----------------
proc Login {s chassis_pass chassis_user console} {

    set pf_flag 1

   puts $s "c_logon $chassis_pass"
   gets $s response
   if {$response == ""} { gets $s response  }

   if {"$response" != "<OK>"}    {set pf_flag 0}
   if {$console == 1} { puts "Logging | $response" }


   puts $s "c_owner $chassis_user"
   gets $s response
   if {$response == ""} { gets $s response  }

   if {$console == 1} { puts "Owner    | $response" }
   if {"$response" != "<OK>"}    {set pf_flag 0}

   return $pf_flag
}
#
# ----------------------------------------
```

AUTOMATING TEST SUITES

Use simple one line shell command to automate a full RFC test based on prebuilt test configuration
TCL uses "EXEC" in order to execute a process



```
C:\Program Files\Xena>Xena2544run_27.exe

XenaRun - Xena Test Configuration Executer
Copyright c Xena Networks 2012 - 2013


ERROR: The option --config is mandatory and must be provided.

Valid options:

  -c, --config         Required. Path to test configuration file.
  -t, --testtype       Test type to execute: { conf (*) | perf } (Y.1564 only).
  -u, --user           Set the username (default: xenacli)
  -r, --reportpath     Path where reports are saved.
  -o, --company        Company name.
  -l, --logo           Path to logo file.
  -h, --help           Display this help screen.

C:\Program Files\Xena>
```

WANT MORE INFORMATION?

RESOURCES

Wiki:	wiki.xenanetworks.com
Website:	www.xenanetworks.com/resources/
Email:	support@xenanetworks.com

XenaWiki

Detailed user manuals and technical documentation