



Xena Scripting

Command-line syntax for controlling Xena testers.

June, 2011

Table of Contents

SCRIPTING OVERVIEW	7
<i>Connecting to the chassis</i>	<i>8</i>
<i>Relation to the Xena Manager</i>	<i>9</i>
<i>Command syntax</i>	<i>10</i>
<i>Status messages</i>	<i>11</i>
<i>Defaults and wild-carding.....</i>	<i>12</i>
<i>Special scripting commands</i>	<i>13</i>
<i>Example input.....</i>	<i>14</i>
<i>Example output</i>	<i>16</i>
CHASSIS PARAMETERS.....	18
C_LOGON <i>PASSWORD</i>	18
C_OWNER <i>USERNAME</i>	18
C_KEEPALIVE <i>TICKS</i>	19
C_RESERVATION <i>WHATTODO</i>	19
C_RESERVEDBY <i>USERNAME</i>	20
C_LOGOFF.....	20
C_DOWN <i>MAGIC WHATODO</i>	20
C_CAPABILITIES <i>INTEGER INTEGER</i>	20
C_MODEL <i>MODEL</i>	21
C_SERIALNO <i>SERIALNO</i>	21
C_VERSIONNO <i>SERVER DRIVER</i>	21
C_PORTCOUNTS <i>PORTCOUNT PORTCOUNT</i> ...	22
C_INFO ?	22
C_ALLPORTCAPS ?	22
C_NAME <i>CHASSISNAME</i>	23
C_COMMENT <i>COMMENT</i>	23
C_PASSWORD <i>PASSWORD</i>	23
C_IPADDRESS <i>ADDRESS SUBNETMASK GATEWAY</i>	24
C_FLASH <i>ONOFF</i>	24
C_CONFIG ?	24
C_INDICES <i>SESSION SESSION</i> ...	25
C_STATSESSION [<i>SES</i>] <i>TYP ADR OWN OPS REQ RSP</i>	25
C_STATS ?	25
C_FILESTART <i>TYPE SIZE TIME MODE CHK NAME</i>	26
C_FILEDATA <i>OFFSET DATABYTES</i>	26
C_FILEFINISH <i>MAGIC</i>	26
MODULE PARAMETERS	28
M_RESERVATION <i>WHATTODO</i>	28

M_RESERVEDBY <i>USERNAME</i>	28
M_MODEL <i>MODEL</i>	29
M_SERIALNO <i>SERIALNO</i>	29
M_VERSIONNO <i>IMAGE</i>	29
M_TIMESYNC <i>MODE</i>	30
M_STATUS <i>TEMPERATURE</i>	30
M_CFPTYPE <i>INFO</i>	30
M_CFPCONFIG <i>PORTS SPEED</i>	31
M_INFO ?	31
M_CONFIG ?.....	31
M_UPGRADE <i>MAGIC IMAGENAME</i>	32
M_UPGRADEPROGRESS <i>PROGRESS</i>	32
PORT PARAMETERS	33
P_RESERVATION <i>WHATTODO</i>	33
P_RESERVEDBY <i>USERNAME</i>	33
P_RESET	34
P_CAPABILITIES <i>INTEGER INTEGER</i>	34
P_INTERFACE <i>INTERFACE</i>	35
P_STATUS <i>OPTICALPOWER</i>	35
P_INFO ?.....	35
P_SPEEDSELECTION <i>SELECTION</i>	36
P_SPEED <i>MBPS</i>	36
P_AUTONEGSELECTION <i>ONOFF</i>	36
P_RECEIVESYNC <i>SYNCSTATUS</i>	37
P_COMMENT <i>COMMENT</i>	37
P_SPEEDREDUCTION <i>PPM</i>	37
P_INTERFRAMEGAP <i>MINBYTES</i>	38
P_MACADDRESS <i>HEXDATA</i>	38
P_IPADDRESS <i>ADDRESS SUBNET GATEWAY WILD</i>	39
P_ARPREPLY <i>ONOFF</i>	39
P_PINGREPLY <i>ONOFF</i>	39
P_PAUSE <i>ONOFF</i>	40
P_FLASH <i>ONOFF</i>	40
P_RANDOMSEED <i>VALUE</i>	40
P_AUTOTRAIN <i>INTERVAL</i>	41
P_LATENCYMODE <i>MODE</i>	41
P_LATENCYOFFSET <i>VALUE</i>	42
P_LOOPBACK <i>LOOPMODE</i>	42
P_CHECKSUM <i>ONOFF</i>	43
P_GAPMONITOR <i>START STOP</i>	43
P_TRAFFIC <i>ONOFF</i>	44

P_CAPTURE <i>ONOFF</i>	44
P_XMITONE <i>HEXDATA</i>	44
P_CONFIG ?.....	45
P_FULLCONFIG ?	45
STREAM PARAMETERS	46
PS_INDICES <i>SID SID ...</i>	46
PS_CREATE [<i>SID</i>].....	46
PS_DELETE [<i>SID</i>]	47
PS_ENABLE [<i>SID</i>] <i>ONOFF</i>	47
PS_PACKETLIMIT [<i>SID</i>] <i>COUNT</i>	47
PS_COMMENT [<i>SID</i>] <i>COMMENT</i>	48
PS_TPLDID [<i>SID</i>] <i>TPLDID</i>	48
PS_INSERTFCS [<i>SID</i>] <i>ONOFF</i>	48
PS_ARPREQUEST [<i>SID</i>] <i>MACADDRESS</i>	49
PS_PINGREQUEST [<i>SID</i>] <i>DELAY TTL</i>	49
PS_RATEFRACTION [<i>SID</i>] <i>FRACTION</i>	50
PS_RATEPPS [<i>SID</i>] <i>PPS</i>	50
PS_RATEL2BPS [<i>SID</i>] <i>BPS</i>	51
PS_RATE [<i>SID</i>] ?	51
PS_BURST [<i>SID</i>] <i>SIZE DENSITY</i>	52
PS_PACKETHEADER [<i>SID</i>] <i>HEXDATA</i>	52
PS_HEADERPROTOCOL [<i>SID</i>] <i>SEGMENT SEGMENT</i>	53
PS_MODIFIERCOUNT [<i>SID</i>] <i>COUNT</i>	54
PS_MODIFIER [<i>SID,MID</i>] <i>POS MASK ACT REP</i>	54
PS_MODIFIERRANGE [<i>SID,MID</i>] <i>START STEP STOP</i>	55
PS_PACKETLENGTH [<i>SID</i>] <i>TYPE MIN MAX</i>	55
PS_PAYLOAD [<i>SID</i>] <i>TYPE HEXDATA</i>	56
PS_CONFIG [<i>SID</i>] ?	56
PS_FULLCONFIG ?	57
PS_INJECTFCSERR [<i>SID</i>]	57
PS_INJECTSEQERR [<i>SID</i>]	57
PS_INJECTMISERR [<i>SID</i>].....	58
PS_INJECTPLDERR [<i>SID</i>].....	58
PS_INJECTTPLDERR [<i>SID</i>].....	59
FILTER AND TERM PARAMETERS.....	60
PM_INDICES <i>MID MID ...</i>	60
PM_CREATE [<i>MID</i>].....	60
PM_DELETE [<i>MID</i>]	61
PM_PROTOCOL [<i>MID</i>] <i>SEGMENT SEGMENT</i>	61
PM_POSITION [<i>MID</i>] <i>BYTEOFFSET</i>	61
PM_MATCH [<i>MID</i>] <i>MASK VALUE</i>	62

PM_CONFIG [MID] ?	62
PM_FULLCONFIG ?	62
PL_INDICES LID LID	63
PL_CREATE [LID]	63
PL_DELETE [LID]	63
PL_LENGTH [LID] TYPE SIZE	64
PL_FULLCONFIG ?	64
PF_INDICES FID FID	64
PF_CREATE [FID]	65
PF_DELETE [FID]	65
PF_ENABLE [FID] ONOFF	65
PF_COMMENT [FID] COMMENT	66
PF_CONDITION [FID] TERMS TERMS	66
PF_CONFIG [FID] ?	67
PF_FULLCONFIG ?	67
CAPTURE PARAMETERS	68
PC_TRIGGER START FILTER1 STOP FILTER2	68
PC_KEEP WHICH INDEX BYTES	69
PC_STATS STATUS PACKETS STARTTIME	69
PC_PACKET [CID] HEXDATA	70
PC_EXTRA [CID] TIME LATENCY IFG LENGTH	70
PC_INFO [CID] ?	70
PC_FULLCONFIG ?	71
STATISTICS PARAMETERS	72
PT_TOTAL BPS PPS BYTES PACKETS	72
PT_NOTPLD BPS PPS BYTES PACKETS	72
PT_EXTRA MISCSTATS... ..	73
PT_STREAM [SID] BPS PPS BYTES PACKETS	73
PT_ALL ?	73
PT_CLEAR	74
PR_TOTAL BPS PPS BYTES PACKETS	74
PR_NOTPLD BPS PPS BYTES PACKETS	74
PR_EXTRA MISCSTATS... ..	75
PR_TPLDS TID TID	75
PR_TPLDTRAFFIC [TID] BPS PPS BYT PAC	75
PR_TPLDERRORS [TID] DUMMY SEQ MIS PLD	76
PR_TPLDLATENCY [TID] MIN AVG MAX	76
PR_FILTER [FID] BPS PPS BYTES PACKETS	77
PR_ALL ?	77
PR_ALLERRORS ?	77
PR_CALIBRATE [TID]	78

PR_CLEAR	78
DATASET PARAMETERS	79
PD_INDICES <i>DID DID</i>	79
PD_CREATE [<i>DID</i>]	79
PD_DELETE [<i>DID</i>].....	80
PD_ENABLE [<i>DID</i>] <i>ONOFF</i>	80
PD_SOURCE [<i>DID</i>] <i>TYPE WHICH ID</i>	80
PD_RANGE [<i>DID</i>] <i>START STEP COUNT</i>	81
PD_SAMPLES [<i>DID</i>] <i>VALUE VALUE</i>	81
PD_CONFIG [<i>DID</i>] ?	82
PD_FULLCONFIG ?.....	82
PD_ALL ?	82
40/100G PARAMETERS.....	83
PP_TXLANECONFIG [<i>PID</i>] <i>VIRTLANE SKEW</i>	83
PP_TXLANEINJECT [<i>PID</i>] <i>TYPE</i>	83
PP_TXPRBSCONFIG [<i>PID</i>] <i>DUMMY ONOFF ERRORS</i>	84
PP_TXERRORRATE <i>RATE</i>	84
PP_TXINJECTONE.....	85
PP_RXLANELOCK [<i>PID</i>] <i>HEADERLOCK ALIGNLOCK</i>	85
PP_RXLANESTATUS [<i>PID</i>] <i>VIRTLANE SKEW</i>	85
PP_RXLANEERRORS [<i>PID</i>] <i>HEADER ALIGN BIP8</i>	86
PP_RXPRBSSTATUS [<i>PID</i>] <i>BYTES ERRORS LOCK</i>	86
PP_RXLASERPOWER <i>NANOWATTS</i>	86
PP_RXCLEAR	87
PP_CONFIG ?.....	87
PP_ALL ?.....	87
PP_ALLERRORS ?	88

Scripting overview

The chassis of the Xena test platform can be controlled in two principle fashions: in a point-and-click interactive style using the Xena Manager application, and in a command-line-interface style using a text-based scripting interface.

This document defines the scripting interface, but assumes that you have a basic familiarity with the chassis functionality gained from using the Xena Manager and reading the Xena System Management Overview document.

The script commands are simple lines of text exchanged between a client and the Xena chassis. An example command to the chassis could be:

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

This goes to module 0, port 5, and sets stream 3's rate to 500000 packets per second. The chassis responds with:

```
<OK>
```

You would query for the current value this way:

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

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

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

This document contains a general overview of the scripting mechanism, followed by reference sections describing each group of scriptable parameters in detail. There are a few hundred parameters in total, but only a handful is required for typical simple tasks.

To set up basic traffic patterns and obtain traffic statistics, use the port parameters (starting with `P_ . . .`), the stream parameters (starting with `PS_ . . .`), and the transmit/receive statistics parameters (starting with `PT_ . . .` and `PR_ . . .`).

At the end of the overview sections there is a complete example of how to use a collection of script commands to define and execute some simple operations on a chassis.

Connecting to the chassis

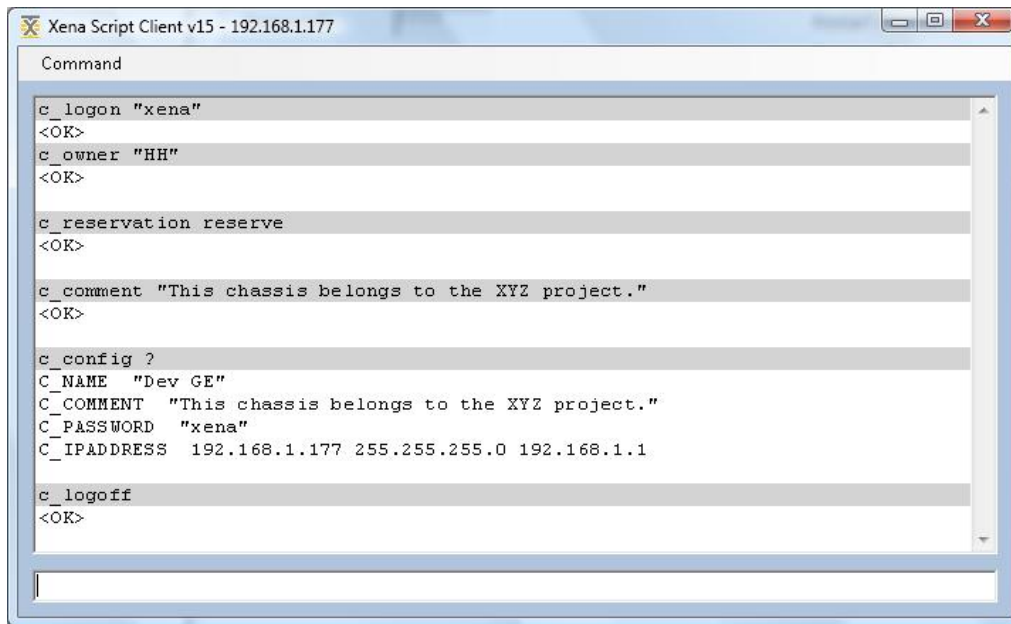
The chassis support multiple concurrent scripting sessions, just like they support multiple concurrent interactive Manager sessions. And like Manager sessions, scripting sessions interact with the chassis in its current state; establishing a scripting session does not in itself impact the chassis state.

In order to start a scripting session, you establish a TCP/IP connection with the chassis using port 22611, on the same IP address as when using the Manager.

You then send lines of ASCII text to the chassis, terminated by CR/LF, and receive lines of ASCII text in response. The first command should be a logon with the valid password for the chassis, or the session will be terminated.

You can use any client platform that is able to establish a TCP/IP connection and send and receive lines of text through it. Typical client platforms include Tcl, Perl, Python, Java, Excel/VBA, and Telnet. You may use client-side functionality to execute script commands conditionally or repetitively.

Xena also provides a simple interactive scripting client application that runs on Windows and allows you to manually type commands to the chassis and see its responses.



```
Command
c_logon "xena"
<OK>
c_owner "HH"
<OK>
c_reservation reserve
<OK>
c_comment "This chassis belongs to the XYZ project."
<OK>
c_config ?
C_NAME "Dev GE"
C_COMMENT "This chassis belongs to the XYZ project."
C_PASSWORD "xena"
C_IPADDRESS 192.168.1.177 255.255.255.0 192.168.1.1
c_logoff
<OK>
```

Figure 1 The Xena Script Client used to change a chassis description.

In order to keep the session alive the client must show some activity every minute or so; else the chassis will assume that the client has stopped, and there are also routers that will kill a TCP session that is inactive for more than a few minutes. The simplest way is to send an empty line, and the chassis will also respond with an empty line.

Chassis resources must as always be reserved before they can be updated, whereas you can view any chassis, module, or port parameter as soon as the session is logged on. Reserving and releasing is done through the `C/M/P_RESERVATION` commands.

Before reservation, a user name must be provided for the scripting session using the `C_OWNER` command. If the chassis has resources reserved to this username they will automatically be granted to this session.

Any line starting with a semicolon is treated as a comment and ignored by the chassis, e.g.:

```
;This script implements the RFC2544 suite
```

Commands to the chassis are not case-sensitive, and replies from the chassis are in upper-case.

Relation to the Xena Manager

Anything you can do with the Xena Manager application you can also do via scripting, and the correspondence is quite straightforward. For example, just as the Manager's PORT PROPERTIES panel has a field for viewing and changing a port's minimum inter-frame gap, the scripting interface can view and change the `P_INTERFRAMEGAP` parameter for doing the same. The same applies to most of the other fields of the Manager's user interface.

However, there are a few areas where the Manager has more advanced functionality which is missing in the scripting interface. This does not limit what you can do, but the way you must do it is more primitive.

- Stream rates and capping. When you specify the rate of a stream using either a percentage, layer-2 Mbps, or packets per second, the Manager calculates the equivalent rates using the other two methods. It also checks that you do not exceed the available bandwidth for the port. This is not available through scripting: you just specify the rate using your method of choice, and you must take care not to exceed the available bandwidth.
- Protocol field editing. The Manager knows the field-by-field layout of various common protocols, and allows you to inspect and edit packet data at the field level. With scripting you just specify packet data as a sequence of hexadecimal bytes.
- Filter conditions. The Manager allows you to enter filter conditions as an easy-to-read Boolean expression on the various terms. With scripting you need to encode the condition using a set of bitmasks.
- Capture protocol decoding. The Manager inspects the raw bytes of captured packets in order to identify the protocols at the header of the packet. With scripting you must decode the packet data yourself if needed.

Also, the manager will disable the user-interface whenever a particular operation is not currently allowed; for instance trying to update the configuration of a port that has not been reserved, changing a parameter for an enabled stream while traffic is on, or changing a filter term used in the condition of an enabled filter. Attempting such things in a scripting session will instead lead to error status messages.

At a more fundamental level, the Manager supports the notion of a testbed containing multiple chassis. This is not applicable through scripting, since each scripting session runs through its own connection to a single chassis, and indeed the chassis are not aware of each other. Any cross-chassis control must be handled at the scripting client environment; in particular cross-chassis statistics such as packet loss.

In contrast, the scripting environment provides wild-carding across modules and ports, which is not available through the Manager.

Command syntax

The scripting language contains similar syntax for setting and getting values of individual parameters of the chassis resources. Some parameters, like inter-frame gap, support both set and get; others, like physical port type, support only get; and a few, like injecting errors, support only set.

You change the value of a settable parameter using:

```
module/port parameter [index,...] value value ...
```

Here, *module* and *port* are the numeric indices for a particular port (for chassis-level parameters neither of these are present, and for module-level parameters only *module* is present); *parameter* is one of the names specified later in this document in the reference sections; *index* is a possible sub-index of the parameter, for instance identifying a stream; and each *value* specifies a value appropriate for the particular parameter. All indices start at zero.

Values are specified using one of the following formats:

- Integer (I): decimal integer, in the 32-bit range, e.g.: 1234567.
- Long (L): decimal integer, in the 64-bit range, e.g.: 123456789123.
- Byte (B): decimal integer, in the 8-bit range, e.g.: 123.
- Hex (H): two hexadecimal digits prefixed by 0x, e.g.: 0xF7.
- String (S): printable 7-bit ASCII characters enclosed in ", e.g.: "A string". Characters with values outside the 32-126 range and the " character itself are specified by their decimal value, outside the quotation marks and separated by commas, e.g.: "A line",13,10,"and the next line".
- Owner (O): a short string used to identify an owner, used for reservation.
- Address (A): a dot-separated IP address, e.g.: 192.168.1.200.

Some parameters allow a variable number of values of a particular type (I*, B*, H*), and these are simply written with spaces in between. For hex values (H*), multiple bytes can be specified using a single 0x prefix, e.g.: 0xF700ABCD2233.

Finally, certain parameters are actually integers, but use coded names for special numeric values to enhance readability, e.g.: (0=OFF , 1=ON).

You query the current value of a gettable parameter using a very similar syntax:

```
module/port parameter [index,...] ?
```

The chassis responds with a line using identical syntax to the change-command, containing the current values. These responses can therefore be 'replayed' back to the chassis to re-establish the value from a previous query. This is actually the core of the load/save mechanism of the Xena Manager, as you can see by using an ordinary text editor to inspect the local files produced by save. You can also change the content if you want to; it is not interpreted by the Xena Manager.

Note that some queries, like P_INFO ? and P_CONFIG ?, are special in that they do not refer to one particular parameter, but rather to a collection of parameters. The response is multiple lines containing the current value of each of these parameters.

Status messages

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

```
<OK>
```

In case something is unacceptable to the chassis, it may return one of the following status messages:

<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.

In case of 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
```

Defaults and wild-carding

The scripting environment provides you with optional default values for the module index and port index, allowing you to change and query parameters without providing the module and port index explicitly.

Default indices are enabled and disabled using the following short commands:

<code>module/port</code>	Set default module and port to the specified values.
<code>port</code>	Set default port to the specified value, retaining the default module.
<code>-/-</code>	Disable the default module and port.
<code>-</code>	Disable the default port, retaining the default module.
<code>module/-</code>	Set the default module, and disable the default port.
<code>?</code>	Show the current default module and port.

When a default module and port is provided, parameters that would otherwise require explicit module and port index values can be written without them, e.g.:

```
PS_RATEPPS [3] 500
^---
#Index error in column 1

0/5
PS_RATEPPS [3] 500
<OK>
```

Replies from the chassis will also use the current default values to suppress the explicit module and port indices when possible.

The scripting environment also provides wild-carding across modules and ports. Using an asterisk as a module or port index effectively makes the chassis execute the command for each value, e.g.:

```
0/* P_INTERFRAMEGAP 30
```

This sets the inter-frame gap for every port on module 0. It will generate an individual status response for each operation, and indeed some may succeed while others fail, for instance due to lack of reservation.

Wild-cards also work for queries. This will give you the inter-frame gap for each port of module 0:

```
0/* P_INTERFRAMEGAP ?
```

Wild-cards cannot be used as default values, but the default and wild-card mechanisms can be combined, for instance to use a default module together with a wild-carded port:

```
0/-  
* P_INTERFRAMEGAP 30
```

Indeed, for chassis with a single module you will typically set it as the default module and then use only port indices.

Special scripting commands

The scripting environment provides a few commands that do not directly interact with the chassis state, but rather support the scripting process itself.

- `SYNC`. This command simply produces a reply of `<SYNC>`, which can be helpful when parsing and delimiting the lines returned from the chassis, in particular when using multi-parameter queries. You can also do `SYNC ON`, which will subsequently cause an automatic `SYNC` after each command. `SYNC OFF` disables this.
- `WAIT n`. This command waits for the specified number of seconds, up to 60, and then produces a reply of `<RESUME>`. This is a simple mechanism for inserting pauses into scripts that are contained in a file and simply sent to the chassis line-by-line. Longer waits and more sophisticated automation require client-side functionality, which must also handle the keep-alives.
- `HELP ?`. This command gives you an overview of the built-in help function, useful when using the scripting environment interactively, as from the Xena Scripting Client.
- `HELP "parameter"`. Gives you a brief overview of the required indices and values for the specified parameter. You are allowed to specify only a prefix of the parameter name, which will then give you the overview for each matching parameter, e.g.: `HELP "P_"` for all port-level parameters. The summary of the required values uses the abbreviations for the various types introduced in the command syntax above, e.g.: `B(0=OFF,1=ON)` which means a single byte value where the two relevant values can be specified using coded names.

Example input

Below is an example of using the scripting commands to define and execute a simple test. A file containing these commands can simply be uploaded to a chassis using the Xena Script Client.

```
; This is an example of using the Xena scripting language to set-up and
; execute a simple test case.
;
; This file is simply sent to TCP/IP port 22611 on a Xena chassis,
; and while it is executing on the chassis it sends lines of text
; back on the same TCP/IP connection.
;
; Much of what you see in response from the chassis is an "<OK>" for
; each new parameter value that you have sent. There will also be a
; blank line in response to each comment you send to the chassis. More
; importantly, of course, you will see the values of the parameters and
; statistics that you explicitly query for.
;
; The chassis has a basic "WAIT" command to allow simple server-side
; waiting. For more advanced scripting logic, you should use a client-
; side scripting environment like Tcl/Perl/Python/Basic/C to send commands
; to the chassis, and retrieve and parse the responses.
;
; The example works on a single port configured in TX-to-RX loop mode
; so that everything sent is also received on the same port.
;
; First we authenticate the connection to the chassis and provide a user
; name for reservation:
C_LOGON "xena"
C_OWNER "example"
; We now set a default port for the session so that all port-specific
; parameters go to this port; this also gives you a single place to edit
; if you want to run the example on a different port. The syntax is
; simply "m/p" where "m" is the module number and "p" is the port number:
0/0
; Let's see what kind of port this is by querying for the interface type:
P_INTERFACE ?
; Now reserve the port, clear any existing configuration, and set it in
; loop-mode:
P_RESERVATION RESERVE
P_RESET
P_LOOPBACK TXON2RX
; Make a stream for transmitting 1000 packets of varying size at a 50% of
; the wire rate for the port. The packet data is just an Ethernet header,
; and we put a modifier on the last byte of the MAC destination address.
; The rest of the packet payload is an incrementing pattern of bytes.
; Finally we insert a Xena test payload at the end containing a TID value
; of 77. We use index 10 for the stream definition itself:
PS_CREATE [10]
PS_COMMENT [10] "Example stream of 1000 packets"
```

```
PS_PACKETLIMIT [10] 1000
PS_PACKETLENGTH [10] RANDOM 100 200
PS_RATEFRACTION [10] 500000
PS_MODIFIERCOUNT [10] 1
PS_MODIFIER [10,0] 5 0xFF000000 DEC 1
PS_PAYLOAD [10] INCREMENTING
PS_TPLDID [10] 77
PS_ENABLE [10] ON

; That was the stream definition. Until now we have been sending values
; to the chassis. Now we'll ask for information from the chassis just to
; verify our configuration. Queries have the same format as used when
; setting values, but with a "?" instead of the values:
PS_PACKETLENGTH [10] ?
P_MACADDRESS ?

; You can also ask for multiple parameters at a time using some special
; pseudo-parameters. Here we'll query for the complete stream definition.
; This will give us all the parameters defined for the stream, including
; some which we have not set explicitly and therefore still have their
; default values from when the configuration was reset:
PS_CONFIG [10] ?

; When parsing the responses from a multi-parameter query you cannot
; immediately tell which parameter value is the last one. To establish a
; fix-point in the stream of response lines you can issue the special "SYNC"
; command which simply responds with "<SYNC>"; so when you receive this
; response you know that there are no more parameters coming:
SYNC

; We're finally ready to run some traffic, but before we start the stream
; we have just defined we'll start the capture function and send out a single
; packet. Since we are in loop mode this packet will be captured on our port,
; and we'll pull it over to the client:
P_CAPTURE ON
P_XMITONE 0x001122334455,AABBCCDDEEFF,2222,FEDCBA9876543210,00000000
PC_STATS ?
PC_PACKET [0] ?

; Ok, now we'll start the stream. Capture is already on. Since this may be a
; slow port we insert a short wait period to make sure all 1000 packets are
; sent, and then we query for the TX and RX statistics:
P_TRAFFIC ON
WAIT 3
PT_ALL ?
PR_ALL ?

; All the packets should have been captured. We pull in a few of them to see
; the varying length and check that the modifier has correctly varied the 5th
; byte. We'll use another multi-parameter query that gives us both the packet
; data and the extra information available for each capture event:
PC_STATS ?
PC_INFO [1] ?
PC_INFO [2] ?
PC_INFO [3] ?
PC_INFO [4] ?
PC_INFO [5] ?

; Even though the single stream of the port has run dry we must still explicitly
; stop traffic generation, and we also stop capturing:
```

```
P_TRAFFIC OFF
P_CAPTURE OFF
; That's it.
; You have now seen how to build a stream, transmit the packets, do some
; capturing, and issue queries for statistics, capture, and configuration.
```

Example output

Below we show the output generated by the chassis when it receives the commands shown in the previous section. A dump like this can be obtained and saved using the Xena Script Client.

You need to do a line-by-line correlation of the two lists in order to fully understand the output. Note that there are sections of blank lines in the output corresponding to the comment lines in the input.

```
<OK>
```

```
<OK>
```

```
P_INTERFACE "SFP-E 10/100/1000M [Auto]"
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
<OK>
```

```
PS_PACKETLENGTH [10] RANDOM 100 200
```

```
P_MACADDRESS 0x02000004EE80
```

```
PS_ENABLE [10] ON
```

```
PS_PACKETLIMIT [10] 1000
```

```
PS_COMMENT [10] "Example stream of 1000 packets"
```

```
PS_RATEFRACTION [10] 500000
```

```
PS_BURST [10] -1 100
```

```
PS_PACKETHEADER [10] 0x00000000000002000004EE800000
```

```
PS_HEADERPROTOCOL [10] ETHERNET
```

```
PS_MODIFIERCOUNT [10] 1
```

```
PS_MODIFIER [10,0] 5 0xFF000000 DEC 1
PS_PACKETLENGTH [10] RANDOM 100 200
PS_PAYLOAD [10] INCREMENTING
PS_TPLDID [10] 77
PS_INSERTFCS [10] ON
```

<SYNC>

<OK>

<OK>

```
PC_STATS 0 1
PC_PACKET [0] 0x001122334455AABBCCDDEEFF2222FEDCBA9876543210F06ECC85
```

<OK>

<RESUME>

```
PT_TOTAL 0 0 149443 1001
PT_NOTPLD 0 0 26 1
PT_STREAM [10] 0 0 149417 1000
P_RECEIVESYNC NO_SYNC
PR_TOTAL 0 0 149443 1001
PR_NOTPLD 0 0 26 1
PR_EXTRA 0 0
PR_TPLDS 77
PR_TPLDTRAFFIC [77] 0 0 149417 1000
PR_TPLDERRORS [77] 0 0 0 0
PR_TPLDLATENCY [77] -1 -1 -1
```

```
PC_STATS 1 102
PC_EXTRA [1] 41037920 0 4677534 185
PC_PACKET [1] 0x00000000000002000004EE800000E0F10111213141516...70337C2E
PC_EXTRA [2] 41040920 0 191 187
PC_PACKET [2] 0x0000000000FF02000004EE800000E0F1011121314151617...4777BAC1
PC_EXTRA [3] 41043584 0 189 143
PC_PACKET [3] 0x0000000000FE02000004EE800000E0F10111213...CF8D7283
PC_EXTRA [4] 41046584 0 191 185
PC_PACKET [4] 0x0000000000FD02000004EE800000E0F10111213141516...3C62BFB5
PC_EXTRA [5] 41048944 0 189 106
PC_PACKET [5] 0x0000000000FC02000004EE800000E0F1011...5BB0875D
```

<OK>

<OK>

Chassis parameters

The chassis parameters correspond to the CHASSIS PROPERTIES panel of the Manager, and deal with basic information about, and configuration of, the chassis itself (rather than its modules and test ports), as well as overall control of the scripting session.

The chassis parameter names all have the form `C_xxx` and use neither a module index nor a port index.

C_LOGON *password*

You log on to the chassis by setting the value of this parameter to the correct password for the chassis. All other commands will fail if the session has not been logged on.

password: string, containing the correct password value.

Summary: set only, value type: S

Example, set:

```
C_LOGON "xena"
```

C_OWNER *username*

Identify the owner of the scripting session. The name can be any short quoted string up to eight characters long. This name will be used when reserving ports prior to updating their configuration.

There is no authentication of the users, and the chassis does not have any actual user accounts. Multiple concurrent connections may use the same owner name, but only one connection can have any particular resource reserved at any given time.

Until an owner is specified the chassis configuration can only be read. Once specified, the session can reserve ports for that owner, and will inherit any existing reservations for that owner retained at the chassis.

username: string, containing the name of the owner of this session.

Summary: set and get, value type: O

Example, set or get:

```
C_OWNER "HH"
```

C_KEEPLIVE *ticks*

You can request this value from the chassis, simply to let it (as well as any routers and proxies between you) know that the connection is still valid.

ticks: integer, an increasing number from the chassis.

Summary: get only, value type: I

Example, set:

```
C_KEEPLIVE 1234
```

C_RESERVATION *whattodo*

You set this parameter to reserve, release, or relinquish the chassis itself. The chassis must be reserved before any of the chassis-level parameters can be changed.

The owner of the session must already have been specified. Reservation will fail if any modules or ports are reserved to other users.

whattodo: coded byte, containing the operation to perform:
RELEASE
RESERVE
RELINQUISH

The reservation parameters are slightly asymmetric with respect to set/get. When querying for the current reservation state, the chassis will use these values:

```
RELEASED  
RESERVED_BY_YOU  
RESERVED_BY_OTHER
```

Summary: set or get, value type: B

Example, set:

```
C_RESERVATION RESERVE
```

Example, get:

```
C_RESERVATION RESERVED_BY_YOU
```

C_RESERVEDBY *username*

Identify the user who has the chassis reserved. The empty string if the chassis is not currently reserved.

username: string, containing the name of the current owner of the chassis.

Summary: get only, value type: O

Example, get:

```
C_RESERVEDBY "HH"
```

C_LOGOFF

Terminates the current scripting session. Courtesy only, the chassis will also handle disconnection at the TCP/IP level.

Summary: set only.

Example, set:

```
C_LOGOFF
```

C_DOWN *magic whatodo*

Shuts down the chassis, and either restarts it in a clean state or leaves it powered off.

magic: integer, must be the special value -1480937026.

whattodo: coded byte, what to do after shutting chassis down:
RESTART
POWEROFF

Summary: set only, value types: I, B

Example, set:

```
C_DOWN -1480937026 RESTART
```

C_CAPABILITIES *integer integer ...*

A series of integer values specifying various internal limits of the chassis.

integer: integer, internally defined limit values.

Summary: get only, value types: I *

Example, get:

```
C_CAPABILITIES 1 25 ...
```

C_MODEL *model*

Obtains the specific model of this Xena chassis.

model: string, the Xena model designation for the chassis.

Summary: get only, value type: S

Example, get:

```
C_MODEL "C1-M2SFP+"
```

C_SERIALNO *serialno*

Obtains the unique serial number of this particular Xena chassis.

serialno: integer, the serial number of this chassis.

Summary: get only, value type: I

Example, get:

```
C_SERIALNO 12345678
```

C_VERSIONNO *server driver*

Obtains the version numbers of the software installed on the chassis.

server: integer, the server version number.

driver: integer, the driver version number.

Summary: get only, value types: I, I

Example, get:

```
C_VERSIONNO 200 30
```

C_PORTCOUNTS *portcount portcount ...*

Obtains the number of ports in each module slot of the chassis, and indirectly the number of slots and modules.

Note: CFP modules return the number 8 which is the maximum number of 10G ports, but the actual number of ports can be configured dynamically using the `M_CFPCONFIG` command.

portcount: byte, the number of ports, typically 2 or 6, or 0 for an empty slot.

Summary: get only, value types: B*

Example, get:

```
C_PORTCOUNTS 2
```

C_INFO ?

Multi-parameter query, obtaining all the non-settable parameters for the chassis.

Summary: get only.

Example, get:

```
C_RESERVATION RESERVED_BY_YOU
C_RESERVEDBY "HH"
C_PORTCOUNTS 2
C_MODEL "C1-M2SFP+"
C_SERIALNO 12345678
C_VERSIONNO 200 30
```

C_ALLPORTCAPS ?

Multi-parameter query, obtaining the port capabilities for all ports of the chassis.

Summary: get only.

Example, get:

```
0/0 P_CAPABILITIES 1110 10000 ...
0/1 P_CAPABILITIES 1000 5000 ...
```

C_NAME *chassisname*

The name of the chassis, as it appears at various places in the Manager user-interface. The name is also used to distinguish the various chassis contained within a testbed and in files containing the configuration for an entire test-case.

chassisname: string, containing the name of the chassis.

Summary: set and get, value type: S

Example, set or get:

```
C_NAME "Lab ABC"
```

C_COMMENT *comment*

The description of the chassis.

comment: string, containing the description of the chassis.

Summary: set and get, value type: S

Example, set or get:

```
C_COMMENT "This chassis belongs to the XYZ project."
```

C_PASSWORD *password*

The password of the chassis, which must be provided when logging on to the chassis.

password: string, containing the password for the chassis.

Summary: set and get, value type: S

Example, set or get:

```
C_PASSWORD "SeCrEt"
```

C_IPADDRESS *address subnetmask gateway*

The network configuration parameters of the chassis management port.

address: address, the static IP address of the chassis.
subnetmask: address, the subnet mask of the local network segment.
gateway: address, the gateway of the local network segment.

Summary: set and get, value types: A, A, A

Example, set or get:

```
C_IPADDRESS 192.168.1.200 255.255.255.0 192.168.1.1
```

C_FLASH *onoff*

Make all the test port LEDs flash on and off with a 1-second interval. This is helpful if you have multiple chassis mounted side by side and you need to identify a specific one.

onoff: coded byte, whether all test port LEDs are blinking:
 OFF
 ON

Summary: set and get, value type: B

Example, set or get:

```
C_FLASH OFF
```

C_CONFIG ?

Multi-parameter query, obtaining all the settable parameters for the chassis.

Summary: get only.

Example, get:

```
C_NAME "Lab ABC"
C_COMMENT "This chassis belongs to the XYZ project."
C_PASSWORD "SeCrEt"
C_IPADDRESS 192.168.1.200 255.255.255.0 192.168.1.1
```

C_INDICES *session session ...*

Obtains the session indices for all current sessions on the chassis.

Summary: get only, value types: I*

Example, get:

```
C_INDICES 0 1 7 9 13
```

C_STATSESSION [*ses*] *typ adr own ops req rsp*

Obtains information and statistics for a particular session on the chassis.

ses: integer, session index.
typ: coded integer, which kind of session:
 MANAGER
 SCRIPT
adr: address, client IP address.
own: string, user name of the session.
ops: long, number of operations done on session.
req: long, number of bytes received by the chassis.
rsp: long, number of bytes sent by the chassis.

Summary: get only, session index, value types: I,A,O,L,L,L

Example, get:

```
C_STATSESSION [7] SCRIPT 88.99.77.66 "HH" 100 12345 23456
```

C_STATS ?

Multi-parameter query, obtaining all the chassis-level statistics.

Summary: get only.

Example, get:

```
C_INDICES 7 13
C_STATSESSION [7] SCRIPT 88.99.77.66 "HH" 100 12345 23456
C_STATSESSION [13] NATIVE 88.99.77.55 "JVN" 111 22345 33456
```

C_FILESTART *type size time mode chk name*

Initiates upload of a file to the chassis. This parameter should be followed by a sequence of C_FILEDATA parameters to provide the file content, and finally a C_FILEFINISH to commit the new file to the chassis.

type: little-endian integer as four hex bytes, the file type, should be 1.
size: little-endian integer as four hex bytes, the number of bytes in the file.
time: little-endian integer as four hex bytes, the Linux date+time of the file.
mode: little-endian integer as four hex bytes, the Linux permissions of the file.
chk: little-endian integer as four hex bytes, the checksum of the file.
name: string, the name and location of the file, as a full path.

Summary: set only, value types: HHHH , HHHH , HHHH , HHHH , HHHH , S

Example, set:

```
C_FILESTART 0x01000000 ... 0xF9B1A073 "/xbin/xenaserver"
```

C_FILEDATA *offset databytes*

Uploads a fragment of a file to the chassis.

offset: integer, the position within the file.
databytes: hex bytes, the data content of a section of the file.

Summary: set only, value types: I , H*

Example, set:

```
C_FILEDATA 10240 0x11FC3344.....43AB
```

C_FILEFINISH *magic*

Completes upload of a file to the chassis. After validation it will replace any existing file with the same name.

magic: integer, must be the special value -1480937026.

Summary: set only, value type: I

Example, set:

C_FILEFINISH -1480937026

Module parameters

The module parameters correspond to the MODULE PROPERTIES panel of the Manager, and deal with basic information about a module itself (rather than its test ports).

The module parameter names all have the form `M_xxx` and require a module index before the parameter name.

M_RESERVATION *whattodo*

You set this parameter to reserve, release, or relinquish a module itself (as opposed to its ports). The module must be reserved before its hardware image can be upgraded.

The owner of the session must already have been specified. Reservation will fail if the chassis or any ports are reserved to other users.

whattodo: coded byte, containing the operation to perform:

- RELEASE
- RESERVE
- RELINQUISH

The reservation parameters are slightly asymmetric with respect to set/get. When querying for the current reservation state, the chassis will use these values:

- RELEASED
- RESERVED_BY_YOU
- RESERVED_BY_OTHER

Summary: set or get, value type: B

Example, set:

```
0 M_RESERVATION RELEASE
```

Example, get:

```
0 M_RESERVATION RELEASED
```

M_RESERVEDBY *username*

Identify the user who has a module reserved. The empty string if the module is not currently reserved.

username: string, containing the name of the current owner of the module.

Summary: get only, value type: O

Example, get:

```
0 M_RESERVEDBY ""
```

M_MODEL *model*

Obtains the specific Xena model of a module.

model: string, the Xena model designation for the module.

Summary: get only, value type: S

Example, get:

```
0 M_MODEL "M2SFP+"
```

M_SERIALNO *serialno*

Obtains the unique serial number of a module.

serialno: integer, the serial number of this module.

Summary: get only, value type: I

Example, get:

```
0 M_SERIALNO 16703
```

M_VERSIONNO *image*

Obtains the version numbers of the hardware image installed on a module.

image: integer, the hardware image version number.

Summary: get only, value types: I

Example, get:

```
0 M_VERSIONNO 170
```

M_TIMESYNC *mode*

Control how the test module time-stamp clock is running, either freely or locked to the chassis system time.

Running with free time allows nano-second precision measurements of latencies, but only when the transmitting and receiving ports are in the same chassis.

Running with locked time enables inter-chassis latency measurements, but can introduce small micro-second level discontinuities as the test module time is adjusted.

mode: coded byte, selecting the time sync mode:
 FREE
 LOCK

Summary: set and get, value type: B

Example, set or get:

```
M_TIMESYNC FREE
```

M_STATUS *temperature*

Get status readings for the test module itself.

temperature: temperature of the main hardware chip, in degrees celcius.

Summary: get only, I*

Example, get:

```
0 M_STATUS 45
```

M_CFPTYPE *info*

Get information about the CFP transceiver currently inserted into the cage of a CFP test module.

info: coded byte, specifying the CFP state:
 NOTCFP (this is not a CFP-based test module)
 NOTPRESENT (no transceiver, the CFP cage is empty)
 NOTFLEXIBLE (transceiver present, supporting a fixed speed and port-count)
 FLEXIBLE (transceiver present, supporting flexible speed and port-count)

Summary: get only, B

Example, get:

```
0 M_CFPTYPE FLEXIBLE
```

M_CFPCONFIG *ports speed*

The current number of ports and their speed of a CFP test module. If the CFP type is NOTFLEXIBLE then it reflects the transceiver currently in the CFP cage. If the CFP type is FLEXIBLE (or NOTPRESENT) then the configuration can be changed explicitly.

The following combinations are possible: 4x10G, 8x10G, 1x40G, 2x40G, and 1x100G.

ports: number of ports.
speed: port speed, in Gbps.

Summary: set and get, value types: B , B

Example, set:

```
0 M_CFPCONFIG 2 40
```

M_INFO ?

Multi-parameter query, obtaining all the non-settable parameters for a module.

Summary: get only.

Example, get:

```
0 M_RESERVATION RESERVED_BY_YOU
0 M_RESERVEDBY "HH"
0 M_MODEL "M2SFP+"
0 M_SERIALNO 16703
0 M_VERSIONNO 170
0 M_STATUS 45
0 M_CFPTYPE NOTCFP
```

M_CONFIG ?

Multi-parameter query, obtaining all the settable parameters for a module.

Summary: get only.

Example, get:

```
0 M_TIMESYNC FREE
```

M_UPGRADE *magic* *imagename*

Transfers a hardware image file from the chassis to a module. This image will take effect when the chassis is powered-on the next time. The transfer takes approximately 3 minutes, but no further action is required by the client.

magic: integer, must be the special value -1480937026.
imagename: string, the fully qualified name of a file previously uploaded to the chassis.

Summary: set only, value types: I, S

Example, set:

```
0 M_UPGRADE -1480937026 "/xbin/xenaimageXE_18"
```

M_UPGRADEPROGRESS *progress*

Provides a value indicating the current stage of an on-going hardware image upgrade operation. This is for information only; the upgrade operation runs to completion by itself.

The progress values are pushed to the client without it having to request them.

progress: integer, the current stage within the three phases: erase, write, verify.

Summary: pushed (get) only, value type: I
 1-100: Erase completion percentage.
 101-200: Write completion percentage + 100.
 201-300: Verify completion percentage + 200.
 0: Failure.

Example, push:

```
0 M_UPGRADEPROGRESS 277
```

Port parameters

The port parameters correspond to the PORT PROPERTIES panel of the Manager, and deal with basic information about, and configuration of, the test ports.

The port parameter names all have the form `P_XXXX` and require both a module index and a port index before the parameter name.

In general, port parameters cannot be changed while traffic is on. Additionally, every stream must be disabled before changing parameters that affect the bandwidth of the port.

P_RESERVATION *whattodo*

You set this parameter to reserve, release, or relinquish a port. The port must be reserved before any of its configuration can be changed, including streams, filters, capture, and datasets.

The owner of the session must already have been specified. Reservation will fail if the chassis or module is reserved to other users.

whattodo: coded byte, containing the operation to perform:
 RELEASE
 RESERVE
 RELINQUISH

The reservation parameters are slightly asymmetric with respect to set/get. When querying for the current reservation state, the chassis will use these values:

```
RELEASED
RESERVED_BY_YOU
RESERVED_BY_OTHER
```

Summary: set or get, value type: B

Example, set:

```
0/5 P_RESERVATION RESERVE
```

Example, get:

```
0/5 P_RESERVATION RESERVED_BY_YOU
```

P_RESERVEDBY *username*

Identify the user who has a port reserved. The empty string if the port is not currently reserved.

Note, that multiple connections can specify the same name with `C_OWNER`, but a resource can only be reserved to one connection. Therefore you cannot count on having the port just because it is reserved in your name. The port is reserved to this connection only if `P_RESERVATION` returns `RESERVED_BY_YOU`.

username: string, containing the name of the current owner of the port.

Summary: get only, value type: O

Example, get:

```
0/5 P_RESERVEDBY "HH"
```

P_RESET

Reset port-level parameters to standard values, and delete all streams, filters, capture, and dataset definitions.

Summary: set only.

Example, set:

```
0/1 P_RESET
```

P_CAPABILITIES *integer integer ...*

A series of integer values specifying various internal limits of a port.

integer: integer, internally defined limit values.

Summary: get only, value types: I*

Example, get:

```
0/1 P_CAPABILITIES 1000 ...
```

P_INTERFACE *interface*

Obtains the physical interface type of a port.

model: string, the name of the port's physical interface.

Summary: get only, value type: S

Example, get:

```
0/5 P_INTERFACE "SFP+ LR"
```

P_STATUS *opticalpower*

Get status readings for the port itself.

opticalpower: integer, received signal level for optical ports, in nanowatts, -1 when not available.

Summary: get only, I*

Example, get:

```
0/5 P_STATUS -1
```

P_INFO ?

Multi-parameter query, obtaining all the non-settable parameters for a port. These parameters should not be included if the port configuration is saved and reloaded at a later time.

Summary: get only.

Example, get:

```
0/5 P_RESERVATION RESERVED_BY_OTHERS
0/5 P_RESERVEDBY "XX"
0/5 P_INTERFACE "SFP+ LR"
0/5 P_SPEED 100
0/5 P_TRAFFIC ON
0/5 P_CAPTURE OFF
```

P_SPEEDSELECTION *selection*

The speed mode for a port with an interface type supporting multiple speeds.

Note: this is only a settable parameter for tri-speed 1G ports. For CFP ports use the `M_CFPCONFIG` command at the module level.

selection: coded byte, containing the speed selection for the port:

- AUTO (auto-negotiate)
- F10M (10 Mbps)
- F100M (100 Mbps)
- F1G (1000 Mbps)
- F10G (10000 Mbps)
- F40G (40000 Mbps)
- F100G (100000 Mbps)

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_SPEEDSELECTION F100M
```

P_SPEED *mbps*

Obtains the current physical speed for a port's interface.

mbps: integer, current speed in units of Mbps.

Summary: get only, value type: I

Example, get:

```
0/0 P_SPEED 100
```

P_AUTONEGSELECTION *onoff*

Whether the port responds to incoming auto-negotiation requests. Only applicable to optical ports, which are fixed speed anyway.

onoff: coded byte, whether the port replies to auto-neg requests.

- OFF
- ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_AUTONEGSELECTION OFF
```

P_RECEIVESYNC *syncstatus*

Obtains the current in-sync status for a port's receive interface.

syncstatus: coded byte, current receive sync status:
NO_SYNC
IN_SYNC

Summary: get only, value type: B

Example, get:

```
0/5 P_RECEIVESYNC IN_SYNC
```

P_COMMENT *comment*

The description of a port.

comment: string, containing the description of the port.

Summary: set and get, value type: S

Example, set or get:

```
0/5 P_COMMENT "This port generates IPTV background traffic."
```

P_SPEEDREDUCTION *ppm*

A speed-reduction applied to the transmit-side of a port, resulting in an effective traffic rate that is slightly lower than the rate of the physical interface.

Speed reduction is effectuated by inserting short idle periods in the generated traffic pattern to consume part of the port's physical bandwidth. The port's clock-speed is not altered.

ppm: integer, specifying the speed reduction in units of parts-per-million.

Summary: set and get, value type: I

Example, set or get:

```
0/3 P_SPEEDREDUCTION 100
```

P_INTERFRAMEGAP *minbytes*

The minimum gap between packets in the traffic generated for a port. The gap includes the Ethernet preamble.

minbytes: integer, specifying the minimum number of byte-times between generated packets.

Summary: set and get, value type: I

Example, set or get:

```
0/3 P_INTERFRAMEGAP 20
```

P_MACADDRESS *hexdata*

A 48-bit Ethernet MAC address specified for a port. This address is used as the default source MAC field in the header of generated traffic for the port, and is also used for support of the ARP protocol.

hexdata: hex bytes, specifying the six bytes of the MAC address.

Summary: set and get, value type: HHHHHH

Example, set or get:

```
0/3 P_MACADDRESS 0x001122AABBCC
```

P_IPADDRESS *address subnet gateway wild*

An IPv4 network configuration specified for a port.

The address is used as the default source address field in the IP header of generated traffic, and the configuration is also used for support of the ARP and PING protocols.

address: address, the IP address of the port.
subnet: address, the subnet mask of the local network segment for the port.
gateway: address, the gateway of the local network segment for the port.
wild: address, wildcards used for ARP and PING replies, must be 255 or 0.

Summary: set and get, value types: A, A, A, A

Example, set or get:

```
0/3 P_IPADDRESS 10.0.0.123 255.255.255.0 10.0.0.1 0.0.0.255
```

P_ARPREPLY *onoff*

Whether the port generates replies using the Address Resolution Protocol.

The port can reply to incoming ARP requests by mapping the IP address specified for the port to the MAC address specified for the port.

ARP reply generation is independent of whether traffic and capture is on for the port.

onoff: coded byte, whether the port replies to ARP requests.
 OFF
 ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_ARPREPLY ON
```

P_PINGREPLY *onoff*

Whether the port generates ping replies using the ICMP protocol.

The port can reply to incoming ping requests to the IP address specified for the port.

Ping reply generation is independent of whether traffic and capture is on for the port.

onoff: coded byte, whether the port replies to ping requests:

OFF
ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_PINGREPLY OFF
```

P_PAUSE *onoff*

Whether a port responds to incoming Ethernet PAUSE frames, by holding back outgoing traffic.

onoff: coded byte, whether PAUSE response is enabled:
OFF
ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_PAUSE OFF
```

P_FLASH *onoff*

Make the test port LED for a particular port flash on and off with a 1-second interval. This is helpful when you need to identify a specific port within a chassis.

onoff: coded byte, whether the test port LED is blinking:
OFF
ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_FLASH ON
```

P_RANDOMSEED *value*

A fixed seed value specified for a port. This value is used for a pseudo-random number generator used when generating traffic that requires random variation in packet length, payload, or modified fields.

As long as no part of the port configuration is changed, the generated traffic patterns are reproducible when restarting traffic for the port. A specified seed value of -1 instead creates variation by using a new time-based seed value each time traffic generation is restarted.

value: integer, specifying a fixed seed value for the pseudo-random number generator.
-1, new random sequence for each start.

Summary: set and get, value type: I

Example, set or get:

```
0/3 P_RANDOMSEED 12345
```

P_AUTOTRAIN *interval*

The interval between sending out training packets, allowing a switch to learn the port's MAC address.

Layer-2 switches configure themselves automatically by detecting the source MAC addresses of packets received on each port.

If a port only receives, and does not itself transmit test traffic, then the switch will never learn its MAC address. Also, if transmission is very rare the switch will age-out the learned MAC address.

By setting the auto-train interval you instruct the port to send switch training packets, independent of whether the port is transmitting test traffic.

interval: integer, specifying the number of seconds between training packets.
0, disable training packets.

Summary: set and get, value type: I

Example, set or get:

```
0/3 P_AUTOTRAIN 60
```

P_LATENCYMODE *mode*

Latency is measured by inserting a time-stamp in each packet when it is transmitted, and relating it to the time when the packet is received.

There are three separate modes for calculating the latency:

- Last-bit-out to last-bit-in, which measures basic bit-transit time, independent of packet length.
- First-bit-out to last-bit-in, which adds the time taken to transmit the packet itself.
- Last-bit-out to first-bit-in, which subtracts the time taken to transmit the packet itself.

The same latency mode must be configured for the transmitting port and the receiving port; otherwise invalid measurements will result.

mode: coded byte, which calculation mode to use:
 LAST2LAST
 FIRST2LAST
 LAST2FIRST

Summary: set and get, value type: B

Example, set or get:

```
0/3 P_LATENCYMODE LAST2LAST
```

P_LATENCYOFFSET *value*

An offset applied to the latency measurements performed for received traffic containing test payloads. This value affects the minimum, average, and maximum latency values obtained through the PR_TPLDLATENCY parameter.

value: integer, specifying the offset for the latency measurements.

Summary: set and get, value type: I

Example, set or get:

```
0/3 P_LATENCYOFFSET 1238
```

P_LOOPBACK *loopmode*

The loop-back mode for a port. Ports can be configured to perform two different kinds of loop-back:

- External RX-to-TX loop-back, where the received packets are re-transmitted immediately. The packets are still processed by the receive logic, and can be captured and analysed.
- Internal TX-to-RX loop-back, where the transmitted packets are received directly by the port itself. This is mainly useful for testing the generated traffic patterns before actual use.

loopmode: coded byte, containing the loop-back mode for the port:
 NONE (normal non-looped operation)
 L1RX2TX (transmit byte-by-byte copy of the incoming packet)
 L2RX2TX (swap source and destination MAC addresses)
 TXON2RX (packet is also transmitted from the port)
 TXOFF2RX (port's transmitter is idle)

Summary: set and get, value type: B

Example, set or get:

```
0/5 P_LOOPBACK L2RX2TX
```

P_CHECKSUM *onoff*

Include an extra payload integrity checksum, which also covers the header protocols following the Ethernet header. It will therefore catch any modifications to the protocol fields (which should therefore not have modifiers on them).

onoff: coded byte, whether the extra checksum is included.
 OFF
 ON

Summary: set and get, value type: B

Example, set or get:

```
0/0 P_CHECKSUM OFF
```

P_GAPMONITOR *start stop*

The gap-start and gap-stop criteria for the port's gap monitor.

The gap monitor expects a steady stream of incoming packets, and detects larger-than-allowed gaps between them. Once a gap event is encountered it requires a certain number of consecutive packets below the threshold to end the event.

start: integer, the maximum allowed gap between packets, in microseconds.
 0, disable gap monitor.
stop: integer, the minimum number of good packets required.
 0, disable gap monitor.

Summary: set and get, value type: I, I

Example, set or get:

```
0/3 P_GAPMONITOR 60000 5
```

P_TRAFFIC *onoff*

Whether a port is transmitting packets. When on, the port generates a sequence of packets with contributions from each stream that is enabled. The streams are configured using the `PS_xxx` parameters.

While traffic is on the streams for this port cannot be enabled or disabled, and the configuration of those streams that are enabled cannot be changed.

onoff: coded byte, whether traffic generation is active for this port:
OFF
ON

Summary: set and get, value type: B

Example, set or get:

```
0/1 P_TRAFFIC ON
```

P_CAPTURE *onoff*

Whether a port is capturing packets. When on, the port retains the received packets and makes them available for inspection. The capture criterias are configured using the `PC_xxx` parameters.

While capture is on the capture parameters cannot be changed.

onoff: coded byte, whether capture is active for this port:
OFF
ON

Summary: set and get, value type: B

Example, set or get:

```
0/1 P_CAPTURE OFF
```

P_XMITONE *hexdata*

Transmits a single packet from a port, independent of the stream definitions, and independent of whether traffic is on. A valid Frame Check Sum is written into the final four bytes.

hexdata: hex bytes, the data content of the packet to be transmitted.

Summary: set only, value types: H*

Example, set:

```
P_XMITONE 0x554433.....48EE
```

P_CONFIG ?

Multi-parameter query, obtaining all the settable parameters for a port itself, but excluding streams, filters, etc.

Summary: get only.

Example, get:

```
0/5 P_SPEEDSELECTION F100M
0/5 P_COMMENT "This port generates IPTV background traffic."
0/5 P_SPEEDREDUCTION 100
0/5 P_INTERFRAMEGAP 20
0/5 P_MACADDRESS 0x001122AABBCC
0/5 P_IPADDRESS 10.0.0.123 255.255.255.0 10.0.0.1 0.0.0.255
0/5 P_ARPREPLY ON
0/5 P_PINGREPLY OFF
0/5 P_PAUSE OFF
0/5 P_RANDOMSEED 12345
0/5 P_LATENCYMODE LAST2LAST
0/5 P_LATENCYOFFSET 1238
0/5 P_LOOPBACK L2RX2TX
```

P_FULLCONFIG ?

Multi-parameter query, obtaining all the settable parameters for a port, including streams, filters, capture, and datasets.

These parameters comprise the complete user-definable configuration for the port.

Summary: get only.

Example, get:

```
0/5 P_SPEEDSELECTION F100M
0/5 P_COMMENT "This port generates IPTV background traffic."
0/5 P_SPEEDREDUCTION 100
0/5 P_INTERFRAMEGAP 20
.
.
.
```

Stream parameters

The stream parameters correspond to the STREAM DEFINITION panel of the Manager, and deal with configuration of the traffic streams transmitted from a port.

Whether the port is actually transmitting packets is specified by the P_TRAFFIC parameter.

While the port is transmitting, the parameters of any enabled stream cannot be changed.

The stream parameter names all have the form PS_XXX and require both a module index and a port index, as well as a sub-index identifying a particular stream.

PS_INDICES *sid sid ...*

The full list of which streams are defined for a port. These are the sub-index values that are used for the parameters defining the traffic patterns transmitted for the port.

Setting the value of this parameter creates a new empty stream for each value that is not already in use, and deletes each stream that is not mentioned in the list. The same can be accomplished one-stream-at-a-time using the PS_CREATE and PS_DELETE commands.

sid: integer, the sub-index of a stream definition for the port.

Summary: set and get, value types: I*

Example, set or get:

```
0/1 PS_INDICES 0 1 5
```

PS_CREATE [*sid*]

Creates an empty stream definition with the specified sub-index value.

sid: integer, the sub-index value of the stream definition to create.

Summary: set only, stream index.

Example, set:

```
0/1 PS_CREATE [5]
```

PS_DELETE [*sid*]

Deletes the stream definition with the specified sub-index value.

sid: integer, the sub-index value of the stream definition to delete.

Summary: set only, stream index.

Example, set:

```
0/1 PS_DELETE [5]
```

PS_ENABLE [*sid*] *onoff*

Whether a stream contributes outgoing packets for a port. The values can be toggled between ON and SUPPRESS while traffic is enabled at the port level.

The sum of the rates of all enabled streams must not exceed the effective port rate.

sid: integer, the sub-index value of the stream definition.

onoff: coded integer, whether the stream is enabled:

```
OFF
ON
SUPPRESS
```

Summary: set and get, stream index, value type: B

Example, set or get:

```
0/1 PS_ENABLE [5] ON
```

PS_PACKETLIMIT [*sid*] *count*

The number of packets that will be transmitted when traffic is started on a port. A value of -1 makes the stream transmit continuously.

sid: integer, the sub-index value of the stream definition.

count: integer, the number of packets.
-1 (disable packet limitation)

Summary: set and get, stream index, value type: I

Example, set or get:

```
0/1 PS_PACKETLIMIT [5] 25
```

PS_COMMENT [*sid*] *comment*

The description of a stream.

sid: integer, the sub-index value of the stream definition.

comment: string, containing the description of the stream.

Summary: set and get, stream index, value type: S

Example, set or get:

```
0/1 PS_COMMENT [5] "Stream for ..."
```

PS_TPLDID [*sid*] *tpldid*

The identifier of the test payloads inserted into packets transmitted for a stream. A value of -1 disables test payloads for the stream.

Test payloads are inserted at the end of each packet, and contains time-stamp and sequence-number information. This allows the receiving port to provide error-checking and latency measurements, in addition to the basic counts and rate measurements provided for all traffic.

The test payload identifier furthermore allows the receiving port to distinguish multiple different streams, which may originate from multiple different chassis. Since test payloads are an inter-port and inter-chassis mechanism, the test payload identifier assignments should be planned globally across all the chassis and ports of the testbed.

sid: integer, the sub-index value of the stream definition.

tpldid: integer, the test payload identifier value.
-1 (disable test payloads)

Summary: set and get, stream index, value type: I

Example, set or get:

```
0/1 PS_TPLDID [5] 17
```

PS_INSERTFCS [*sid*] *onoff*

Whether a valid frame checksum is added to the packets of a stream.

sid: integer, the sub-index value of the stream definition.

onoff: coded integer, whether frame checksums are inserted:
OFF
ON

Summary: set and get, stream index, value type: B

Example, set or get:

```
0/1 PS_INSERTFCS [5] ON
```

PS_ARPREQUEST [sid] macaddress

Generates an outgoing ARP request on the test port.

The packet header for the stream must contain an IP protocol segment, and the destination IP address is used in the ARP request. If there is a gateway IP address specified for the port and it is on a different subnet than the destination IP address in the packet header, then the gateway IP address is used instead.

The framing of the ARP request matches the packet header, including any VLAN protocol segments.

This script parameter does not generate an immediate result, but waits until an ARP reply is received on the test port. If no reply is received within a short timeout it returns <FAILED>.

macaddress: hex bytes, specifying the six bytes of the MAC address.

Summary: get only, value type: HHHHHH

Example, get:

```
0/3 PS_ARPREQUEST [5] 0x001122776655
```

PS_PINGREQUEST [sid] delay ttl

Generates an outgoing ping request using the ICMP protocol on the test port.

The packet header for the stream must contain an IP protocol segment, with valid source and destination IP addresses.

The framing of the ping request matches the packet header, including any VLAN protocol segments, and the destination MAC address must also be valid, possibly containing a value obtained with PS_ARPREQUEST.

This script parameter does not generate an immediate result, but waits until a ping reply is received on the test port. If no reply is received within a short timeout it returns <FAILED>.

delay: integer, the number of milliseconds for the ping reply to arrive.

ttl: byte, the time-to-live value in the ping reply packet.

Summary: get only, value type: I, B

Example, get:

```
0/3 PS_PINGREQUEST [5] 12 128
```

PS_RATEFRACTION [*sid*] *fraction*

The rate of the traffic transmitted for a stream, expressed in millionths of the effective rate for the port.

The bandwidth consumption includes the inter-frame gap, and is independent of the length of the packets generated for the stream.

The sum of the bandwidth consumption for all the enabled streams must not exceed the effective rate for the port.

Setting this parameter also instructs the Manager to attempt to keep the rate-percentage unchanged in case it has to cap stream rates. Getting it is only valid if the rate was last set using this parameter.

sid: integer, the sub-index value of the stream definition.

fraction: integer, stream rate expressed as a value between 0..1000000.

Summary: set and get, stream index, value type: I

Example, set or get:

```
0/1 PS_RATEFRACTION [5] 500000
```

PS_RATEPPS [*sid*] *pps*

The rate of the traffic transmitted for a stream, expressed in packets per second.

The bandwidth consumption is heavily dependent on the length of the packets generated for the stream, and also on the inter-frame gap for the port.

The sum of the bandwidth consumption for all the enabled streams must not exceed the effective rate for the port.

Setting this parameter also instructs the Manager to attempt to keep the packets-per-second unchanged in case it has to cap stream rates. Getting it is only valid if the rate was last set using this parameter.

sid: integer, the sub-index value of the stream definition.

pps: integer, stream rate expressed as packets per second.

Summary: set and get, stream index, value type: I

Example, set or get:

```
0/1 PS_RATEPPS [5] 300000
```

PS_RATEL2BPS [sid] bps

The rate of the traffic transmitted for a stream, expressed in units of bits-per-second at layer-2, thus including the Ethernet header but excluding the inter-frame gap.

The bandwidth consumption is somewhat dependent on the length of the packets generated for the stream, and also on the inter-frame gap for the port.

The sum of the bandwidth consumption for all the enabled streams must not exceed the effective rate for the port.

Setting this parameter also instructs the Manager to attempt to keep the layer-2 bps rate unchanged in case it has to cap stream rates. Getting it is only valid if the rate was last set using this parameter.

sid: integer, the sub-index value of the stream definition.

bps: long integer, stream rate expressed as deka-bits-per-second.

Summary: set and get, stream index, value type: L

Example, set or get:

```
0/1 PS_RATEL2BPS [5] 800000000
```

PS_RATE [sid] ?

Query the rate of the traffic transmitted for a stream in the manner it was last expressed. The response is one of PS_RATEFRACTION, PS_RATEPPS, or PS_RATEL2BPS.

This is the rate that is highlighted in the Manager, and the one it attempts to keep unchanged in case it has to cap stream rates.

sid: integer, the sub-index value of the stream definition.

Summary: get only, stream index.

Example, get:

```
0/1 PS_RATE [5] ?
```

PS_BURST [*sid*] *size density*

The burstiness of the traffic transmitted for a stream, expressed in terms of the number of packets in each burst, and how densely they are packed together.

The burstiness does not affect the bandwidth consumed by the stream, only the spacing between the packets.

A density value of 100 means that the packets are packed tightly together, only spaced by the minimum inter-frame gap. A value of 0 means even, non-bursty, spacing. The exact spacing achieved depends on the other enabled streams of the port.

sid: integer, the sub-index value of the stream definition.

size: integer, the number of packets lumped together in a burst.

density: integer, the percentage of the available spacing that is inserted between bursts.

Summary: set and get, stream index, value types: I, I

Example, set or get:

```
0/1 PS_BURST [5] 4 50
```

PS_PACKETHEADER [*sid*] *hexdata*

The first portion of the packet bytes that are transmitted for a stream. This starts with the 14 bytes of the Ethernet header, followed by any contained protocol segments.

All packets transmitted for the stream start with this fixed header. Individual byte positions of the packet header may be varied on a packet-to-packet basis using modifiers.

The full packet comprises the header, the payload, an optional test payload, and the frame checksum.

The header data is specified as raw bytes, since the script environment does not know the field-by-field layout of the various protocol segments. But the Manager does, so in practise you may use the Manager's packet editor, and simply query for the resulting hex string at the script level.

sid: integer, the sub-index value of the stream definition.

hexdata: hex bytes, the raw bytes comprising the packet header.

Summary: set and get, stream index, value types: H*

Example, set or get:

```
0/1 PS_PACKETHEADER [5] 0x02AABB...
```

PS_HEADERPROTOCOL [sid] segment segment ...

The protocol segments corresponding to the packet header bytes of a stream.

This is mainly for information purposes, and the stream will transmit the packet header bytes even if no protocol segments are specified.

However, some protocols, in particular IPv4, require a packet-by-packet fix-up of various header fields, in particular length and checksum fields. These fix-ups are applied according to the protocol segments specified for the header.

<i>sid</i> :	integer, the sub-index value of the stream definition.	
<i>segment</i> :	coded integer, a number specifying a built-in protocol segment:	
	ETHERNET	(14 bytes)
	VLAN	(4 bytes)
	ARP	(28 bytes)
	IP	(20 bytes)
	IPV6	(40 bytes)
	UDP	(8 bytes)
	TCP	(20 bytes)
	LLC	(3 bytes)
	SNAP	(5 bytes)
	GTP	(20 bytes)
	ICMP	(8 bytes)
	RTP	(12 bytes)
	RTCP	(4 bytes)
	STP	(35 bytes)
	SCTP	(12 bytes)
	MACCTRL	(4 bytes)
	MPLS	(4 bytes)
	PBBTAG	(4 bytes)
	FCOE	(14 bytes)
	FC	(24 bytes)
	FCOETAIL	(4 bytes)
	IGMP0	(12 bytes)
	IGMP1	(16 bytes)
	-n	(n bytes raw segment)

Summary: set and get, stream index, value types: B*

Example, set or get:

```
0/1 PS_HEADERPROTOCOL [5] ETHERNET -4 IP UDP
```

PS_MODIFIERCOUNT [*sid*] *count*

The number of modifiers active on the packet header of a stream.

Each modifier is specified using PS_MODIFIER.

sid: integer, the sub-index value of the stream definition.

count: integer, the number of modifiers for the stream.

Summary: set and get, stream index, value type: I

Example, set or get:

```
0/1 PS_MODIFIERCOUNT [5] 1
```

PS_MODIFIER [*sid,mid*] *pos mask act rep*

A packet modifier for a stream header. The headers of each packet transmitted for the stream will be varied according to the modifier specification.

This parameter requires two sub-indices, one for the stream and one for the modifier.

A modifier is positioned at a fixed place in the header, selects a number of consecutive bits starting from that position, and applies an action to those bits in each packet. Packets can be repeated so that a certain number of identical packets are transmitted before applying the next modification.

sid: integer, the sub-index value of the stream definition.

mid: integer, the sub-index value of the modifier.

pos: integer, the byte position from the start of the packet.

mask: four hex bytes, the mask specifying which bits to affect.

act: coded integer, which action to perform on the affected bits:

INC (increment)

DEC (decrement)

RANDOM (random)

rep: integer, how many times to repeat each packet.

Summary: set and get, stream index, modifier index, value types: I , HHHH , I , I

Example, set or get:

```
0/1 PS_MODIFIER [5,0] 6 0x0FC00000 RANDOM 1
```

PS_MODIFIERRANGE [*sid,mid*] *start step stop*

Range specification for a packet modifier for a stream header, specifying which values the modifier should take on.

This applies only to incrementing and decrementing modifiers; random modifiers always produce every possible bit pattern.

The range is specified as a three values: start, step, and stop, where stop must be equal to start plus a multiple of step.

sid: integer, the sub-index value of the stream definition.
mid: integer, the sub-index value of the modifier.
start: integer, the minimum modifier value.
step: integer, the increment between modifier values.
stop: integer, the maximum modifier value.

Summary: set and get, stream index, modifier index, value types: I, I, I

Example, set or get:

```
0/1 PS_MODIFIERRANGE [5,0] 100 10 200
```

PS_PACKETLENGTH [*sid*] *type min max*

The length distribution of the packets transmitted for a stream.

The length of the packets transmitted for a stream can be varied from packet to packet, according to a choice of distributions within a specified min..max range.

The length of each packet is reflected in the size of the payload portion of the packet, whereas the header has constant length.

Length variation complements, and is independent of, the content variation produced by header modifiers.

sid: integer, the sub-index value of the stream definition.
type: coded integer, the kind of distribution:
 FIXED (all packets have min size)
 INCREMENTING (incrementing from min to max)
 BUTTERFLY (min, max, min+1, max-1, min+2, max-2, etc)
 RANDOM (random between min and max)
 MIX (a mixture of sizes between 56 and 1518, average 464 bytes)
min: integer, lower limit on the packet length.
max: integer, upper limit on the packet length.

Summary: set and get, stream index, value types: I, I, I

Example, set or get:

```
0/1 PS_PACKETLENGTH [5] BUTTERFLY 100 1500
```

PS_PAYLOAD [sid] type hexdata

The payload content of the packets transmitted for a stream.

The payload portion of a packet starts after the header and continues up until the test payload or the frame checksum.

The payload may vary in length, and is filled with either an incrementing sequence of byte values, or a repeated multi-byte pattern.

sid: integer, the sub-index value of the stream definition.
type: coded integer, the kind of payload content:
 PATTERN (a pattern is repeated up through the packet)
 INCREMENTING (bytes are incremented up through the packet)
 PRBS (bytes are randomized from packet to packet)
hexdata: hex bytes, a pattern of bytes to be repeated.

Summary: set and get, stream index, value types: B, H*

Example, set or get:

```
0/1 PS_PAYLOAD [5] PATTERN 0xAABB00FFEE
```

PS_CONFIG [sid] ?

Multi-parameter query, obtaining all the parameters for a specific stream.

sid: integer, the sub-index value of the stream definition.

Summary: get only, stream index.

Example, get:

```
0/1 PS_ENABLE [5] ON
0/1 PS_PACKETLIMIT [5] 25
.
.
.
0/1 PS_PAYLOAD [5] PATTERN 0xAABB00FFEE
```

PS_FULLCONFIG ?

Multi-parameter query, obtaining all parameters for all streams defined on a port.

Summary: get only.

Example, get:

```
0/1 PS_INDICES 0 1 5
0/1 PS_ENABLE [0] ON
0/1 PS_PACKETLIMIT [0] 25
.
.
0/1 PS_PAYLOAD [0] PATTERN 0xAABB00FFEE
0/1 PS_ENABLE [1] OFF
.
.
0/1 PS_ENABLE [5] ON
.
.
```

PS_INJECTFCSERR [*sid*]

Force a frame checksum error in one of the packets currently being transmitted from a stream.

This can aid in analysing the error-detection functionality of the system under test.

Traffic must be on for the port, and the stream must be enabled.

sid: integer, the sub-index value of the stream definition.

Summary: set only, stream index.

Example, set:

```
0/1 PS_INJECTFCSERR [5]
```

PS_INJECTSEQERR [*sid*]

Force a sequence error by skipping a test payload sequence number in one of the packets currently being transmitted from a stream.

This can aid in analysing the error-detection functionality of the system under test.

Traffic must be on for the port, and the stream must be enabled and include test payloads.

sid: integer, the sub-index value of the stream definition.

Summary: set only, stream index.

Example, set:

```
0/1 PS_INJECSEQSERR [5]
```

PS_INJECTMISERR [*sid*]

Force a disorder error by swapping the test payload sequence numbers in two of the packets currently being transmitted from a stream.

This can aid in analysing the error-detection functionality of the system under test.

Traffic must be on for the port, and the stream must be enabled and include test payloads.

sid: integer, the sub-index value of the stream definition.

Summary: set only, stream index.

Example, set:

```
0/1 PS_INJECTMISERR [5]
```

PS_INJECTPLDERR [*sid*]

Force a payload integrity error in one of the packets currently being transmitted from a stream.

Payload integrity validation is only available for incrementing payloads, and the error is created by changing a byte from the incrementing sequence.

The packet will have a correct frame checksum, but the receiving Xena chassis will detect the invalid payload based on information in the test payload.

Traffic must be on for the port, and the stream must be enabled and include test payloads.

sid: integer, the sub-index value of the stream definition.

Summary: set only, stream index.

Example, set:

```
0/1 PS_INJECTPLDERR [5]
```

PS_INJECTTPLDERR [*sid*]

Force a test payload error in one of the packets currently being transmitted from a stream.

This means that the test payload will not be recognized at the receiving port, so it will be counted as a no-test-payload packet, and there will be a lost packet for the stream.

Traffic must be on for the port, and the stream must be enabled and include test payloads.

sid: integer, the sub-index value of the stream definition.

Summary: set only, stream index.

Example, set:

```
0/1 PS_INJECTTPLDERR [5]
```

Filter and term parameters

The filter and term parameters correspond to the FILTER TERMS and FILTER DEFINITION panels of the Manager, and deal with configuration of the basic terms and conditions active on the received traffic of a port.

The filter and term parameter names all have the form `PM_XXX`, `PL_XXX` and `PF_XXX` and require both a module index and a port index, as well as a sub-index identifying a particular match term, length term, or filter.

The match terms and length terms provide basic true/false indications for each packet received on the port, and each filter specifies a compound Boolean condition on these true/false values to determine if the filter as a whole is true/false.

While a filter is enabled, neither its condition nor the definition of each match term or length term used by the condition can be changed.

PM_INDICES *mid mid ...*

The full list of which match terms are defined for a port. These are the sub-index values that are used for the parameters defining the content-based matching of packets received for the port.

Setting the value of this parameter creates a new empty match term for each value that is not already in use, and deletes each match term that is not mentioned in the list. The same can be accomplished one-match-term-at-a-time using the `PM_CREATE` and `PM_DELETE` commands.

mid: integer, the sub-index of a match term definition for the port.

Summary: set and get, value types: I*

Example, set or get:

```
0/1 PM_INDICES 1 3
```

PM_CREATE [*mid*]

Creates an empty match term definition with the specified sub-index value.

mid: integer, the sub-index value of the match term definition to create.

Summary: set only, match term index.

Example, set:

```
0/1 PM_CREATE [3]
```

PM_DELETE [*mid*]

Deletes the match term definition with the specified sub-index value.

A match term cannot be deleted while it is used in the condition of any filter for the port.

mid: integer, the sub-index value of the match term definition to delete.

Summary: set only, match term index.

Example, set:

```
0/1 PM_DELETE [3]
```

PM_PROTOCOL [*mid*] *segment segment ...*

The protocol segments assumed on the packets received on the port.

This is mainly for information purposes, and helps you identify which portion of the packet header is being matched. The actual value definition the match position is specified with `PM_POSITION`.

mid: integer, the sub-index value of the match term definition.

segment: coded integer, a number specifying a built-in protocol segment:
Uses the same coded values as the `PS_HEADERPROTOCOL` parameter.

Summary: set and get, match term index, value types: B*

Example, set or get:

```
0/1 PM_PROTOCOL [3] ETHERNET VLAN
```

PM_POSITION [*mid*] *byteoffset*

The position within each received packet where content matching begins for the port.

mid: integer, the sub-index value of the match term definition.

byteoffset: integer, offset from the start of the packet bytes.

Summary: set and get, match term index, value types: I

Example, set or get:

```
0/1 PM_POSITION [3] 14
```

PM_MATCH [mid] mask value

The value that must be found at the match term position for packets received on the port. The mask can make certain bit positions don't-care.

mid: integer, the sub-index value of the match term definition.
mask: eight hex bytes, which bits are significant in the match operation.
value: eight hex bytes, the value that must be found for the match term to be true.

Summary: set and get, match term index, value types: HHHHHHHH , HHHHHHHH

Example, set or get:

```
0/1 PM_MATCH [3] 0x0FFF000000000000 0x0123000000000000
```

PM_CONFIG [mid] ?

Multi-parameter query, obtaining all the parameters for a specific match term.

mid: integer, the sub-index value of the match term definition.

Summary: get only, match term index.

Example, get:

```
0/1 PM_PROTOCOL [3] ETHERNET VLAN
0/1 PM_POSITION [3] 14
0/1 PM_MATCH [3] 0x0FFF000000000000 0x0123000000000000
```

PM_FULLCONFIG ?

Multi-parameter query, obtaining all parameters for all match terms defined on a port.

Summary: get only.

Example, get:

```
0/1 PM_INDICES 1 3
0/1 PM_PROTOCOL [1] ETHERNET IP UDP
.
.
0/1 PM_PROTOCOL [3] ETHERNET VLAN
0/1 PM_POSITION [3] 14
```

```
0/1 PM_MATCH [3] 0x0FFF000000000000 0x0123000000000000
```

PL_INDICES *lid lid ...*

The full list of which length terms are defined for a port. These are the sub-index values that are used for the parameters defining the length-based matching of packets received for the port.

Setting the value of this parameter creates a new empty length term for each value that is not already in use, and deletes each length term that is not mentioned in the list. The same can be accomplished one-length-term-at-a-time using the `PL_CREATE` and `PL_DELETE` commands.

lid: integer, the sub-index of a length term definition for the port.

Summary: set and get, value types: I*

Example, set or get:

```
0/1 PL_INDICES 0
```

PL_CREATE [*lid*]

Creates an empty length term definition with the specified sub-index value.

lid: integer, the sub-index value of the length term definition to create.

Summary: set only, length term index.

Example, set:

```
0/1 PL_CREATE [0]
```

PL_DELETE [*lid*]

Deletes the length term definition with the specified sub-index value.

A length term cannot be deleted while it is used in the condition of any filter for the port.

lid: integer, the sub-index value of the length term definition to delete.

Summary: set only, length term index.

Example, set:

```
0/1 PL_DELETE [0]
```

PL_LENGTH [lid] type size

The specification for a length-based check that is applied on the packets received on the port.

lid: integer, the sub-index value of the length term definition.
type: coded integer, whether to test for shorter-than or longer-than:
 AT_MOST (packet length must be less-than-or-equal to specified size)
 AT_LEAST (packet length must be greater-than-or-equal to specified size)
size: integer, the value to compare the packet length against.

Summary: set and get, length term index, value types: I, I

Example, set or get:

```
0/1 PL_LENGTH [0] AT_MOST 100
```

PL_FULLCONFIG ?

Multi-parameter query, obtaining all parameters for all length terms defined on a port.

Summary: get only.

Example, get:

```
0/1 PL_INDICES 0
0/1 PL_LENGTH [0] AT_MOST 100
```

PF_INDICES fid fid ...

The full list of which filters are defined for a port. These are the sub-index values that are used for the parameters defining the compound conditions on the match/length terms operating on the packets received for the port.

Setting the value of this parameter creates a new empty filter for each value that is not already in use, and deletes each filter that is not mentioned in the list. The same can be accomplished one-filter-at-a-time using the PF_CREATE and PF_DELETE commands.

fid: integer, the sub-index of a filter definition for the port.

Summary: set and get, value types: I*

Example, set or get:

```
0/1 PF_INDICES 0 3
```

PF_CREATE [*fid*]

Creates an empty filter definition with the specified sub-index value.

fid: integer, the sub-index value of the filter definition to create.

Summary: set only, filter index.

Example, set:

```
0/1 PF_CREATE [3]
```

PF_DELETE [*fid*]

Deletes the filter definition with the specified sub-index value.

fid: integer, the sub-index value of the filter definition to delete.

Summary: set only, filter index.

Example, set:

```
0/1 PF_DELETE [3]
```

PF_ENABLE [*fid*] *onoff*

Whether a filter is currently active on a port.

While a filter is enabled its condition cannot be changed, nor can any match term or length terms used by it.

fid: integer, the sub-index value of the filter definition.

onoff: coded integer, whether the filter is enabled:
OFF
ON

Summary: set and get, filter index, value types: B

Example, set or get:

```
0/1 PF_ENABLE [3] OFF
```

PF_COMMENT [*fid*] *comment*

The description of a filter.

fid: integer, the sub-index value of the filter definition.
comment: string, containing the description of the filter.

Summary: set and get, filter index, value types: S

Example, set or get:

```
0/1 PF_COMMENT [3] "Filter for ..."
```

PF_CONDITION [*fid*] *terms terms ...*

The boolean condition on the terms specifying when the filter is satisfied. The condition uses a canonical *and-or-not* expression on the match terms and length terms.

The condition is specified using a number of compound terms, each encoded as an integer value specifying an arbitrary set of the match terms and length terms defined for the port. Each match/length term has a specific power-of-two value, and the set is encoded as the sum of the values for the contained terms:

Value for match term [mid] = 2^{mid}
 Value for length term [lid] = $2^{(\text{lid}+16)}$

A compound term is true if all the match terms and length terms contained in it are true. This supports the *and*-part of the condition.

If some compound term is satisfied, the condition as a whole is true. This is the *or*-part of the condition.

The first few compound terms at the even positions (second, fourth, ...) are inverted, and all the contained match terms and length terms must be false at the same time that the those of the preceding compound term are true. This is the *not*-part of the condition.

In practise, the simplest way to generate these encodings is to use the Manager, which supports Boolean expressions using the operators &, |, and ~, and simply query the chassis for the resulting script-level definition.

fid: integer, the sub-index value of the filter definition.
terms: integer, encoding a compound term which is a set of the match terms and length terms.

Summary: set and get, filter index, value types: I*

Example, set or get:

```
0/1 PF_CONDITION [3]
```

PF_CONFIG [fid] ?

Multi-parameter query, obtaining all the parameters for a specific filter.

fid: integer, the sub-index value of the filter definition.

Summary: get only, filter index.

Example, get:

```
0/1 PF_COMMENT [3] "Filter for ..."  
0/1 PF_CONDITION [3]  
0/1 PF_ENABLE [3] OFF
```

PF_FULLCONFIG ?

Multi-parameter query, obtaining all parameters for all filters defined on a port.

Summary: get only.

Example, get:

```
0/1 PF_INDICES 0 3  
0/1 PF_COMMENT [0] "Filter for ..."  
.  
.  
0/1 PF_COMMENT [3] "Filter for ..."  
0/1 PF_CONDITION [3]  
0/1 PF_ENABLE [3] OFF
```

Capture parameters

The capture parameters correspond to the CAPTURE CONTROL, CAPTURE RESULTS and CAPTURE GRAPH panels of the Manager, and deal with configuration of the capture criteria and inspection of the captured data from a port.

Whether the port is enabled for capturing packets is specified by the `P_CAPTURE` parameter. Captured packets are indexed starting from 0, and are stored in a buffer that is cleared before capture starts.

While on, the capture configuration parameters cannot be changed.

The capture parameter names all have the form `PC_XXXX` and require both a module index and a port index. The per-packet parameters also use a sub-index identifying a particular packet in the capture buffer.

`PC_TRIGGER start filter1 stop filter2`

The criteria for when to start and stop the capture process for a port

Even when capture is enabled with `P_CAPTURE`, the actual capturing of packets can be delayed until a particular start criteria is met by a received packet.

Likewise, a stop criteria can be specified, based on a received packet. If no explicit stop criteria is specified capture stops when the internal buffer runs full.

In buffer overflow situations, if there is an explicit stop criteria then the latest packets will be retained (and the early ones discarded), and otherwise the earliest packets are retained (and the later ones discarded).

start: coded integer, the criteria for starting the actual packet capture:
 ON (start immediately when capture is started)
 FCSERR (start when receiving a packet containing a frame checksum error)
 FILTER (start when receiving a packet satisfying a filter condition)

filter1: integer, the index of a particular filter for the start criteria.

stop: coded integer, the criteria for stopping the actual packet capture:
 FULL (continue until the capture buffer runs full)
 FCSERR (continue until receiving a packet with a frame checksum error)
 FILTER (continue until receiving a packet satisfying a filter condition)

filter2: integer, the index of a particular filter for the stop criteria.

Summary: set and get, value types: I, I, I, I

Example, set or get:

```
0/1 PC_TRIGGER FILTER 3 FULL 0
```

PC_KEEP *which index bytes*

Which packets to keep once the start criteria has been triggered for a port. Also how big a portion of each packet to retain, saving space for more packets in the capture buffer.

which: coded integer, which general kind of packets to keep:
 ALL (keep all packets between the start and stop trigger)
 FCSERR (keep only those packets with frame checksum errors)
 NOTPLD (keep only those packets without a test payload)
 TPLD (keep only those packets with a test payload and specific id)
 FILTER (keep only those packets satisfying a specific filter condition)

index: integer, test payload id or filter index for which packets to keep.

bytes: integer, how many bytes to keep in the buffer for of each packet.
 -1, no limit on packet size.

Summary: set and get, value types: I, I, I

Example, set or get:

```
0/1 PC_KEEP TPLD 17 30
```

PC_STATS *status packets starttime*

Obtains the number of packets currently in the capture buffer for a port. The count is reset to zero when capture is turned on.

status: long integer, 1 if capture has been stopped because of overflow, 0 if still running.
packets: long integer, the number of packets in the buffer.
starttime: long integer, time when capture was started, in nano seconds since midnight 1/1/2010.

Summary: get only, value types: L, L, L

Example, get:

```
0/1 PC_STATS 0 987 3453543453
```

PC_PACKET [cid] hexdata

Obtains the raw bytes of a captured packet for a port.

The packet data may be truncated if the `PC_KEEP` parameter specified a limit on the number of bytes kept.

cid: integer, the sub-index value of the captured packet.
hexdata: hex bytes, the raw bytes kept for the packet.

Summary: get only, capture packet index, value types: H*

Example, get:

```
0/1 PC_PACKET [986] 0x00AA00CC...
```

PC_EXTRA [cid] time latency ifg length

Obtains extra information about a captured packet for a port. The information comprises time of capture, latency, inter-frame gap, and original packet length.

Latency is only valid for packets with test payloads and where the originating port is on the same module and therefore has the same clock.

cid: integer, the sub-index value of the captured packet.
time: long integer, the number of nano-seconds since capture was started.
latency: long integer, the number of nano-seconds since packet was transmitted.
ifg: long integer, the number of byte-times since previous packet.
length: integer, the real length of the packet on the wire.

Summary: get, capture packet index, value types: L, L, L, I

Example, get:

```
0/1 PC_EXTRA [986] 30000000 40000 100 555
```

PC_INFO [cid] ?

Multi-parameter query, obtaining all the information for a particular captured packet for a port.

cid: integer, the sub-index value of the captured packet.

Summary: get only, capture packet index.

Example, get:

```
0/1 PC_PACKET [986] 0x00AA00CC...
0/1 PC_EXTRA [986] 30000000 40000 100 555
```

PC_FULLCONFIG ?

Multi-parameter query, obtaining all parameters of the capture configuration for a port. This does not include any captured data itself.

Summary: get only.

Example, get:

```
0/1 PC_TRIGGER FILTER 3 FULL 0
0/1 PC_KEEP TPLD 17 30
```

Statistics parameters

The statistics parameters correspond to the TRANSMIT STATISTICS and RECEIVE STATISTICS panels of the Manager, and provide quantitative information about the transmitted and received packets on a port.

The statistics parameter names all have the form `PT_XXX` and `PR_XXX` and require both a module index and a port index. Those parameters dealing with a specific transmitted stream also have a sub-index, and so have the parameters dealing with a specific received test payload id and a specific filter.

All bit- and byte-level statistics are at layer-2, so they include the full Ethernet frame, and exclude the inter-frame gap and preamble.

`PT_TOTAL` *bps* *pps* *bytes* *packets*

Obtains statistics concerning all the packets transmitted on a port.

bps: long integer, number of bits transmitted in the last second.
pps: long integer, number of packets transmitted in the last second.
bytes: long integer, number of bytes transmitted since statistics were cleared.
packets: long integer, number of packets transmitted since statistics were cleared.

Summary: get only, value types: L, L, L, L

Example, get:

```
0/0 PT_TOTAL 8000000000 15000000 12345678987654 123456789876
```

`PT_NOTPLD` *bps* *pps* *bytes* *packets*

Obtains statistics concerning the packets without a test payload transmitted on a port.

bps: long integer, number of bits transmitted in the last second.
pps: long integer, number of packets transmitted in the last second.
bytes: long integer, number of bytes transmitted since statistics were cleared.
packets: long integer, number of packets transmitted since statistics were cleared.

Summary: get only, value types: L, L, L, L

Example, get:

```
0/0 PT_NOTPLD 800000 1000 1234567 12345
```

PT_EXTRA miscstats..

Obtains statistics concerning special packets transmitted on a port since transmit statistics were cleared.

arprequests: long integer, number of ARP request packets transmitted.
arpreplies: long integer, number of ARP reply packets transmitted.
pingrequests: long integer, number of PING request packets transmitted.
pingreplies: long integer, number of PING reply packets transmitted.
injectedfcs: long integer, number of injected packets with FCS errors.
injectedseq: long integer, number of injected packets with sequence errors.
injectedmis: long integer, number of injected packets with disorder errors.
injectedint: long integer, number of injected packets with integrity errors.
injectedtid: long integer, number of injected packets with TID errors.
training: long integer, number of MAC training packets transmitted.

Summary: get only, value types: L*

Example, get:

```
0/0 PT_EXTRA 0 0 0 0 0 0 0 0 0 0 0
```

PT_STREAM [sid] bps pps bytes packets

Obtains statistics concerning the packets of a specific stream transmitted on a port.

sid: integer, the sub-index value of the stream definition.
bps: long integer, number of bits transmitted in the last second.
pps: long integer, number of packets transmitted in the last second.
bytes: long integer, number of bytes transmitted since statistics were cleared.
packets: long integer, number of packets transmitted since statistics were cleared.

Summary: get only, stream index, value types: L, L, L, L

Example, get:

```
0/0 PT_STREAM [5] 800000000 1500000 1234567898765 12345678987
```

PT_ALL ?

Multi-parameter query, obtaining all the transmit statistics for a port.

Summary: get only.

Example, get:

```
0/0 PT_TOTAL 8000000000 15000000 12345678987654 123456789876
```

```

0/0 PT_NOTPLD 800000 1000 1234567 12345
0/0 PT_STREAM [0] 800000000 1500000 1234567898765 12345678987
0/0 PT_STREAM [1] 700000000 1300000 1134567898765 11345678987
0/0 PT_STREAM [5] 600000000 1100000 1034567898765 10345678987

```

PT_CLEAR

Clear all the transmit statistics for a port. The byte and packet counts will restart at zero.

Summary: set only.

Example, set:

```
0/0 PT_CLEAR
```

PR_TOTAL *bps pps bytes packets*

Obtains statistics concerning all the packets received on a port.

bps: long integer, number of bits received in the last second.

pps: long integer, number of packets received in the last second.

bytes: long integer, number of bytes received since statistics were cleared.

packets: long integer, number of packets received since statistics were cleared.

Summary: get only, value types: L, L, L, L

Example, get:

```
0/0 PR_TOTAL 8000000000 15000000 12345678987654 123456789876
```

PR_NOTPLD *bps pps bytes packets*

Obtains statistics concerning the packets without a test payload received on a port.

bps: long integer, number of bits received in the last second.

pps: long integer, number of packets received in the last second.

bytes: long integer, number of bytes received since statistics were cleared.

packets: long integer, number of packets received since statistics were cleared.

Summary: get only, value types: L, L, L, L

Example, get:

```
0/0 PR_NOTPLD 800000 1000 1234567 12345
```

PR_EXTRA miscstats...

Obtains statistics concerning special packets received on a port since receive statistics were cleared.

fcerrors: long integer, number of packets with frame checksum errors.
pauseframes: long integer, number of Ethernet pause frames.
arprequests: long integer, number of ARP request packets received.
arpreplies: long integer, number of ARP reply packets received.
pingrequests: long integer, number of PING request packets received.
pingreplies: long integer, number of PING reply packets received.
gapcount: long integer, number of gap monitor gaps encountered.
gapduration: long integer, combined duration of gap monitor gaps encountered, microseconds.

Summary: get only, value types: L*

Example, get:

```
0/0 PR_EXTRA 0 123 0 0 0 0
```

PR_TPLDS tid tid ...

Obtain the set of test payload ids observed among the received packets since receive statistics were cleared. Traffic statistics for these test payload streams will have non-zero byte and packet count.

tid: integer, identifier of test payload with non-zero packet count.

Summary: get only, value types: I*

Example, get:

```
0/0 PR_TPLDS 17 77
```

PR_TPLDTRAFFIC [tid] bps pps byt pac

Obtains traffic statistics concerning the packets with a particular test payload id received on a port.

tid: integer, the identifier of the test payload.
bps: long integer, number of bits received in the last second.
pps: long integer, number of packets received in the last second.
byt: long integer, number of bytes received since statistics were cleared.
pac: long integer, number of packets received since statistics were cleared.

Summary: get only, test payload id, value types: L, L, L, L

Example, get:

```
0/0 PR_TPLDTRAFFIC [17] 80000000 150000 123456789876 1234567898
```

PR_TPLDERRORS [tid] dummy seq mis pld

Obtains statistics concerning errors in the packets with a particular test payload id received on a port.

The error information is derived from analysing the various fields contained in the embedded test payloads of the received packets, independent of which chassis and port may have originated the packets.

Note that packet-lost statistics involve both a transmitting port and a receiving port, and in particular knowing which port originated the packets with a particular test payload identifier. This information requires knowledge of the global test environment, and is not supported at the port-level.

tid: integer, the identifier of the test payload.
dummy: long integer, not used.
seq: long integer, number of non-incrementing-sequence-number events.
mis: long integer, number of swapped-sequence-number disorder events.
pld: long integer, number of packets with non-incrementing payload content.

Summary: get only, test payload id, value types: L, L, L, L

Example, get:

```
0/0 PR_TPLDERRORS [17] 0 1 0 0
```

PR_TPLDLATENCY [tid] min avg max

Obtains statistics concerning the latency experienced by the packets with a particular test payload id received on a port. The values are adjusted by the port-level P_LATENCYOFFSET value.

A special value of -1 is returned if latency numbers are not applicable.

Latency is only meaningful when the clocks of the transmitter and receiver are synchronized. This requires the two ports to be on the same test module, and it requires knowledge of the global test environment to ensure that packets are in fact routed between these ports.

tid: integer, the identifier of the test payload.
min: long integer, nanoseconds, minimum latency for test payload stream.
avg: long integer, nanoseconds, average latency for test payload stream.
max: long integer, nanoseconds, maximum latency for test payload stream.

Summary: get only, test payload id, value types: L, L, L

Example, get:

```
0/0 PR_TPLDLATENCY [17] 2400 2900 3700
```

PR_FILTER [fid] bps pps bytes packets

Obtains statistics concerning the packets satisfying the condition of a particular filter for a port.

fid: integer, the sub-index of the filter definition.
bps: long integer, number of bits received in the last second.
pps: long integer, number of packets received in the last second.
bytes: long integer, number of bytes received since statistics were cleared.
packets: long integer, number of packets received since statistics were cleared.

Summary: get only, filter index, value types: L,L,L,L

Example, get:

```
0/0 PR_FILTER [3] 80000000 150000 123456789876 1234567898
```

PR_ALL ?

Multi-parameter query, obtaining all the receive statistics for a port.

Summary: get only.

Example, get:

```
0/0 PR_TOTAL 8000000000 15000000 12345678987654 123456789876
0/0 PR_NOTPLD 800000 1000 1234567 12345
0/0 PR_EXTRA 0 123 0 0 0 0 0
0/0 PR_TPLDS 17 77
0/0 PR_TPLDTRAFFIC [17] 80000000 150000 123456789876 1234567898
0/0 PR_TPLDERRORS [17] 0 1 0 0
0/0 PR_TPLDLATENCY [17] 2400 2900 3700
0/0 PR_TPLDTRAFFIC [77] 80000000 150000 123456789876 1234567898
.
.
0/0 PR_FILTER [3] 80000000 150000 123456789876 1234567898
```

PR_ALLERRORS ?

Multi-parameter query, obtaining all the test payload id error statistics for a port.

Summary: get only.

Example, get:

```
0/0 PR_TPLDS 17 77
0/0 PR_TPLDERRORS [17] 0 1 0 0
0/0 PR_TPLDERRORS [77] 2 3 0 7
```

PR_CALIBRATE [*tid*]

Calibrate the latency calculation for packets containing a particular test payload id received on a port. The current minimum value will be set as the new base.

tid: integer, the identifier of the test payload.

Summary: set only, test payload id.

Example, get:

```
0/0 PR_CALIBRATE [17]
```

PR_CLEAR

Clear all the receive statistics for a port. The byte and packet counts will restart at zero.

Summary: set only.

Example, set:

```
0/0 PR_CLEAR
```

Dataset parameters

The dataset parameters correspond to the HISTOGRAM CONTROL and HISTOGRAM RESULTS panels of the Manager, and deal with configuration of data collection and retrieval of samples from a port.

The dataset parameter names all have the form `PD_xxx` and require both a module index and a port index, as well as a sub-index identifying a particular dataset.

A dataset has a number of 'buckets' and counts the packets transmitted or received on a port, possibly limited to those with a particular test payload id. The packet length, inter-frame gap preceding it, or its latency is measured, and the bucket whose range contains this value is incremented.

While a dataset is actively collecting samples its parameters cannot be changed.

PD_INDICES *did* *did* ...

The full list of which datasets are defined for a port. These are the sub-index values that are used for the parameters defining which datasets are collected based on the packets transmitted or received on a port.

Setting the value of this parameter creates a new dataset for each value that is not already in use, and deletes each dataset that is not mentioned in the list. The same can be accomplished one-filter-at-a-time using the `PD_CREATE` and `PD_DELETE` commands.

did: integer, the sub-index of a dataset definition for the port.

Summary: set and get, value types: I*

Example, set or get:

```
0/1 PD_INDICES 0 1
```

PD_CREATE [*did*]

Creates a dataset definition with the specified sub-index value.

did: integer, the sub-index value of the dataset definition to create.

Summary: set only, dataset index.

Example, set:

```
0/1 PD_CREATE [0]
```

PD_DELETE [did]

Deletes the dataset definition with the specified sub-index value.

did: integer, the sub-index value of the dataset definition to delete.

Summary: set only, dataset index.

Example, set:

```
0/1 PD_DELETE [0]
```

PD_ENABLE [did] onoff

Whether a dataset is currently active on a port.

When turned on, all the bucket counts are cleared to zero. Subsequently each packet matching the dataset source criteria is counted into one of the buckets.

While a dataset is enabled its parameters cannot be changed.

did: integer, the sub-index value of the dataset definition.

onoff: coded integer, whether the dataset is enabled:
OFF
ON

Summary: set and get, dataset index, value types: B

Example, set or get:

```
0/1 PD_ENABLE [0] ON
```

PD_SOURCE [did] type which id

The source criteria specifying what is counted, and for which packets, by a dataset of a port.

did: integer, the sub-index value of the dataset definition.

type: coded integer, specifying what is counted and for which packets.
TXIFG (inter-frame gap of transmitted packets, measured in bytes)
TXLEN (length of transmitted packets, measured in bytes)
RXIFG (inter-frame gap of received packets, measured in bytes)
RXLEN (length of received packets, measured in bytes)
RXLAT (latency of received packets, measured in nanoseconds)

which: coded integer, specifying a further detail on which packets to count:

ALL (all packets specified by the type)
 TPLD (only those packets with a particular test payload)
 FILTER (only those packets satisfying a particular filter)
id: integer, test payload id of filter id for wanted packets.

Summary: set and get, dataset index, value types: I , I , I

Example, set or get:

```
0/1 PD_SOURCE [0] RXLEN TPLD 17
```

PD_RANGE [did] start step count

The bucket ranges used for classifying the packets counted by a dataset of a port.

The packets are either counted by length, measured in bytes, by inter-frame gap to the preceding packet, also measured in bytes, or by latency in transmission measured in nanoseconds.

There are a fixed number of buckets, each middle bucket covering a fixed-size range of values which is a power of two. The first and last buckets count all the packets that don't fit within the ranges of the middle buckets. The buckets are placed at a certain offset by specifying the first value that should be counted by the first middle bucket.

did: integer, the sub-index value of the dataset definition.
start: integer, first value going into the second bucket.
step: integer, the span of each middle bucket:
 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 (bytes, non-latency datasets)
 16, 32, 64, 128, ..., 1048576, 2097152 (nanoseconds, latency datasets)
count: integer, the total number of buckets.

Summary: set and get, dataset index, value types: I , I , I

Example, set or get:

```
0/1 PD_RANGE [0] 100 8 256
```

PD_SAMPLES [did] value value ...

The current set of counts collected by a dataset for a port.

There is one value for each bucket, but any trailing zeros are left out. The list is empty if all counts are zero.

did: integer, the sub-index value of the dataset definition.
value: long integer, the number of packets counted for a bucket.

Summary: get only, dataset index, value types: L*

Example, get:

```
0/1 PD_SAMPLES [0] 342567809534 756767654 3124532463 687
```

PD_CONFIG [did] ?

Multi-parameter query, obtaining all the parameters for a specific dataset.

did: integer, the sub-index value of the dataset definition.

Summary: get only, dataset index.

Example, get:

```
0/1 PD_ENABLE [0] ON
0/1 PD_SOURCE [0] RXLEN 17
0/1 PD_RANGE [0] 100 8 256
```

PD_FULLCONFIG ?

Multi-parameter query, obtaining all parameters for all datasets defined on a port.

Summary: get only.

Example, get:

```
0/1 PD_INDICES 0 1
0/1 PD_ENABLE [0] ON
0/1 PD_SOURCE [0] RXLEN TPLD 17
0/1 PD_RANGE [0] 100 8 256
.
.
0/1 PD_RANGE [1] 0 4 256
```

PD_ALL ?

Multi-parameter query, obtaining all the samples for all datasets defined for a port.

Summary: get only.

Example, get:

```
0/1 PD_SAMPLES [0] 342567809534 756767654 3124532463 687
0/1 PD_SAMPLES [1] 2567809534 6767654 24532463 7 9 13
```

40/100G parameters

The 40/100G parameters provide configuration and status for the CAUI physical coding sub-layer used by 40G and 100G ports.

The data is broken down into a number of lower-speed lanes. For 40G there are 4 lanes of 10 Gbps each. For 100G there are 20 lanes of 5 Gbps each. Within each lane the data is broken down into 66-bit code-words.

During transport the lanes may be swapped and skewed with respect to each other. To deal with this each lane contains marker words with a virtual lane index. The parameters are index with a physical lane index which corresponds to a fixed numbering of the underlying fibers or wavelengths.

The lanes can also be put into PRBS mode where they transmit a bit pattern used for diagnosing fiber-level problems, and the receiving side can lock to these patterns.

Errors can be injected both at the CAUI level and at the bit level.

The 40/100G parameter names all have the form `PP_XXX` and require a module index and a port index, and most also require a physical lane index.

PP_TXLANECONFIG [*pid*] *virtlane* *skew*

The virtual lane index and artificial skew for data transmitted on a specified physical lane.

pid: integer, the lane sub-index.
virtlane: integer, the logical lane number.
skew: integer, the inserted skew on the lane, in bit units.

Summary: set and get, lane index, value types: I, I

Example, set or get:

```
0/1 PP_TXLANECONFIG [19] 17 0
```

PP_TXLANEINJECT [*pid*] *type*

Inject a particular kind of CAUI error into a specific physical lane.

type: coded byte, specifying what kind of error to inject.

HEADERERROR (error in the 2-bit header of a code-word)
 ALIGNERROR (error in the special alignment code-word)
 BIP8ERROR (error in the BIP8 checksum of the alignment code-word)

Summary: set only, lane index, value type: B

Example, set:

```
0/1 PP_TXLANEINJECT [19] ALIGNERROR
```

PP_TXPRBSCONFIG [*pid*] *dummy onoff errors*

The PRBS configuration for a particular lane. When PRBS is enabled for any lane then the overall link is compromised and drops out of sync.

pid: integer, the lane sub-index.
dummy: integer, not used.
onoff: coded integer, whether this lane is transmitting PRBS data.
 OFF
 ON
errors: coded integer, whether bit-level errors are injected into this lane.
 OFF
 ON

Summary: set and get, lane index, value types: I, B, B

Example, set or get:

```
0/1 PP_TXPRBSCONFIG [19] 0 ON OFF
```

PP_TXERRORRATE *rate*

The rate of continuous bit-level error injection. Errors are injected evenly across the lanes where injection is enabled.

rate: long integer, the number of bits between each error.
 0, no error injection.

Summary: set and get, value type: L

Example, set or get:

```
0/1 PP_TXERRORRATE 1000000000
```

PP_TXINJECTONE

Inject a single bit-level error into one of the lanes where injection is enabled.

Summary: set only.

Example, set:

```
0/1 PP_TXINJECTONE
```

PP_RXLANELOCK [*pid*] *headerlock* *alignlock*

Whether the receiver has achieved header lock and alignment lock on the data received on a specified physical lane.

pid: integer, the lane sub-index.

headerlock: coded byte, whether this lane has achieved header lock.
 HEADEROFF
 HEADERON

alignlock: coded byte, whether this lane has achieved alignment lock.
 ALIGNOFF
 ALIGNON

Summary: get only, lane index, value types: B, B

Example, get:

```
0/1 PP_RXLANELOCK [19] HEADERON ALIGNON
```

PP_RXLANESTATUS [*pid*] *virtlane* *skew*

The virtual lane index and actual skew for data received on a specified physical lane. This is only meaningful when the lane is in header lock and alignment lock.

pid: integer, the lane sub-index.

virtlane: integer, the logical lane number.

skew: integer, the measured skew on the lane, in bit units.

Summary: get only, lane index, value types: I, I

Example, get:

```
0/1 PP_RXLANESTATUS [19] 17 66
```

PP_RXLANEERRORS [*pid*] *header align bip8*

Statistics about errors detected at the physical coding sub-layer on the data received on a specified physical lane.

pid: integer, the lane sub-index.
header: long integer, the number of header errors.
align: long integer, the number of alignment errors.
bip8: long integer, the number of bip8 errors.

Summary: get only, lane index, value types: L, L, L

Example, get:

```
0/1 PP_RXLANEERRORS [19] 0 0 5
```

PP_RXPRBSSTATUS [*pid*] *bytes errors lock*

Statistics about PRBS pattern detection on the data received on a specified physical lane.

pid: integer, the lane sub-index.
bytes: long integer, the number of bytes received while in PRBS lock.
errors: long integer, the number of errors detected while in PRBS lock.
lock: coded byte, whether this lane is in PRBS lock.
 PRBSOFF
 PRBSON

Summary: get only, lane index, value types: L, L, B

Example, get:

```
0/1 PP_RXPRBSSTATUS [19] 1000000000 3 ON
```

PP_RXLASERPOWER *nanowatts...*

Reading of the optical power level of the received signal. There is one value for each laser/wavelength, and the number of these depends on the kind of CFP transceiver used. The list is empty if the CFP transceiver does not support optical power read-out.

nanowatts: integer, received signal level, in nanowatts.
 0, when no signal.

Summary: get only, value types: I*

Example, get:

```
0/1 PP_RXLASERPOWER 1000123 1123000 999000 1000987
```

PP_RXCLEAR

Clear all the 40/100G receive statistics for a port.

Summary: set only.

Example, set:

```
0/1 PP_RXCLEAR
```

PP_CONFIG ?

Multi-parameter query, obtaining all the 40/100G parameters for a port.

Summary: get only.

Example, get:

```
0/1 PP_TXLANECONFIG [0] 1 0
0/1 PP_TXPRBSCONFIG [0] 0 ON OFF
0/1 PP_TXLANECONFIG [1] 0 0
.
.
0/1 PP_TXERRORRATE 1000000000
```

PP_ALL ?

Multi-parameter query, obtaining all the 40/100G statistics for a port.

Summary: get only.

Example, get:

```
0/1 PP_RXLANELOCK [0] HEADERON ALIGNON
0/1 PP_RXLANESTATUS [0] 17 66
0/1 PP_RXLANEERRORS [0] 0 0 5
0/1 PP_RXPRBSSTATUS [0] 1000000000 1 ON
.
.
0/1 PP_RXPRBSSTATUS [19] 1000000000 3 ON
```

PP_ALLERRORS ?

Multi-parameter query, obtaining all the 40/100G PRBS error statistics for a port.

Summary: get only.

Example, get:

```
0/1  PP_RXPRBSSTATUS  [0]  1000000000  1  ON
0/1  PP_RXPRBSSTATUS  [1]  1000000000  6  ON
.
.
0/1  PP_RXPRBSSTATUS  [19] 1000000000  3  ON
```