TELEDYNE LECROY
Everywhereyoulook™

Xena OpenAutomation CLI Command Documentation

Release 97.0

Teledyne LeCroy Xena

Apr 24, 2024

TABLE OF CONTENTS

This documentation provides a comprehensive guide to using the *Xena OpenAutomation CLI*, including installation instructions, usage examples, and a detailed reference guide to all available commands. The documentation is organized by command and includes detailed descriptions of each command, along with usage examples and expected output.

In addition to the command reference, the Xena Networks website also provides other resources for learning about and using the XOA platform, including user guides, tutorials, and code examples. The Xena Networks community forum is also a valuable resource for getting help and support with the XOA CLI and other Xena products.

Overall, the XOA CLI command documentation provides a wealth of information for network engineers and developers who are interested in managing Xena test equipment and automating network testing tasks using the command line. With its comprehensive command reference and usage examples, the documentation can help users get up and running quickly and efficiently with the XOA CLI.

The target audience of this document is test specialists who develop and run automated test scripts using Xena test equipment. Users of this document should have the following knowledge and experience:

- Familiarity with the operating system of the development environment.
- Familiarity with Xena test equipment.
- Working knowledge of data communications theory and practice.

Important: To learn *XOA Python API*, go to Xena OpenAutomation Python API Documentation.

CHAPTER

ONE

INTRODUCTION

XOA CLI is a command-line interface for managing Xena Networks test equipment and automating network testing tasks. The XOA CLI is part of the Xena OpenAutomation, which provides a framework for automating network testing tasks using Xena test equipment.

Any client platform can be used to establish a TCP/IP connection and send and receive CLI commands as lines of text. Typical client platforms include Tcl, Perl, Python, BASH, Ruby, Java, and VBA. All Xena chassis support multiple concurrent scripting sessions, enabling different users to work on the same Xena chassis simultaneously.

To start a scripting session simply open a TCP/IP connection to the Xena chassis using TCP port 22611, on the same IP address as when using the Managers. You can then send lines of ASCII text to the chassis (in the XOA CLI command syntax format), terminated by CR/LF, and receive lines of ASCII text in response (also in the XOA CLI command syntax format).

You can either open the scripting connection from a console tool such as **Telnet**, or from the XenaScriptClient application, or the built-in script client bundled with the ValkyrieManager. Then you can interact with the Xena chassis using the XOA CLI command syntax format.

Overall, the XOA CLI is a valuable tool for network engineers and developers who need a flexible and powerful way to manage Xena test equipment and automate network testing tasks. With its simple syntax and comprehensive command set, the XOA CLI can help users streamline their testing workflows and improve the accuracy and reliability of their test results.

Advantages

- Versatile
  - Can be used with ANY scripting and program language
  - Can run in ANY operating system
  - No need to install drivers or proprietary programs
  - Re-use your existing automation framework
- Fast
  - Extremely low communication overhead
  - Very simple and efficient protocol
- Simple

- Purely text based
- No binary modules
- Easy to learn
- Well documented
- Easy to debug
- Reuse existing port configurations

1.1 Ways to Control Xena Testers

Xena test equipment can be controlled in different ways:

1. In a point-and-click interactive style using ValkyrieManager and VulcanManager.
2. In a command-line-interface style using Xena OpenAutomation CLI.
3. In an object-oriented programming style using Xena OpenAutomation Python API.

Note: Using XOA Python API for test automation development is out of the scope of this document. You can go to Xena OpenAutomation Python API Documentation to know more about it.

As an alternative to using the ValkyrieManager, you can interact with the testers using XOA CLI commands. This also allows the tester to be controlled from an scripting environment, and be part of a larger automation environment.

1.2 CLI to Chassis

The XOA CLI allows users to interact with Xena test equipment from the command line, using a set of commands and parameters that can be used to automate a variety of testing tasks. The XOA CLI is designed to be user-friendly and easy to use, with a simple syntax and intuitive command structure.

Some of the tasks that can be performed with the XOA CLI include configuring test equipment, creating and executing test scenarios, generating traffic, and performing detailed analysis of network performance and behavior. The XOA CLI can also be used to automate repetitive tasks, reducing the time and effort required to perform complex network testing tasks.

XOA CLI covers most of the functionalities of ValkyrieManager

---

Note: Please keep this in mind even though most of the examples use the ValkyrieManager for illustration.

CHAPTER

TWO

BASICS

This section helps you understand the basics of XOA CLI.

As an alternative to using the application ValkyrieManager, you can interact with the testers using XOA CLI. This also allows the tester to be controlled from a scripting environment, and be part of a larger automation environment.

XOA CLI is a command-line-interface scripting command which supports multiple concurrent scripting sessions. This makes it easy for different users in different locations to work on the same Xena tester simultaneously.

There are 700+ scriptable commands: from basic streams and capture setup to wild-carding across modules and ports. It is, of course, possible to use the client-side functionality to execute script commands both conditionally and repetitively, which offers real advantages when it comes to test automation.

2.1 Command Syntax

The CLI commands are simple lines of text exchanged between a client and a Xena tester. An example command to the chassis is:

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

This goes to Module 0, Port 5, and sets stream #3's rate to 500,000 frames per second, and the tester responds with:

```
<OK>
```

You would query for the current parameter this way:

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

And the tester would respond in exactly the same way that you set the parameter yourself:

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

Note: ValkyrieManager saves test port configurations (.xpc file) in the exact same CLI command format as used by XOA CLI. This makes it very easy to go back and forth between a ValkyrieManager environment and a XOA CLI environment. For example, exporting a port configuration from ValkyrieCLIManager generates a configuration file in a simple text format that can be edited using a text editing tool such as Microsoft Notepad. It can then be imported back into ValkyrieManager.

The seamless interaction between port configuration files and XOA CLI accelerates your scripting learning curve, letting you get more done quicker as complex test port configurations can easily be defined in ValkyrieManager, and then exported to a text based configuration file, which in turn can be cut & pasted into your scripting tool environment.

In the following sections, you will find the basics of Xena's scripting mechanism, followed by reference sections describing each group of scriptable commands in detail. There are a few hundred commands in total, but only a handful is required for typical simple tasks.

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

In the last section *Sample XOA CLI Script*, you will find an example of how to use a collection of XOA CLI commands to define and execute some simple operations on a Valkyrie tester.

The XOA CLI has similar syntax for setting and getting parameters of individual commands of the chassis resources. Some commands, such as inter-frame gap, support both `set` and `get` actions; others, like physical port type, support only `get`; and a few, like injecting errors, support only `set`.

2.1.1 `set` Syntax

You change parameters or states of the test resources using:

```
<module-index>/<port-index> <command> [<indices>] <parameters>
```

- <module-index> and <port-index> are the numeric sub-indices for locating a particular module or a port.
  - For chassis-level commands, neither of these are present.
  - For module-level commands only <module-index> is present.
  - Both <module-index> and <port-index> are 8-bit long.
  - The value range of <module-index> and <port-index> is 0 - 255.
- <command> is one of the names specified later in this document in *CLI Reference*.
- <indices> are possible sub-indices of the command, e.g. identifying a stream, a Serdes, a modifier, etc.

  - <indices> must be placed inside [ ].
  - Multiple <indices> must be separated by comma, e.g. [0, 1, 2].
  - All <indices> start at zero.
  - All <indices> are 32-bit long, but it doesn't mean they can be any value between 0 and 2^32-1. Usually the maximum value of an index is limited by the capabilities of the chassis/module/port, for example the number of streams on a port, or the number of modifiers on a stream.
- <parameters> specify the values appropriate for the particular command.
  - Multiple parameters are separated by space.
  - A parameter is specified using one of the formats defined in *Parameter Data Types*.

Examples:

```
C_OWNER "username"
0 M_COMMENT "module description"
0/5 P_RESERVATION RESERVE
0/5 PS_RATEPPS [3] 500000
```

2.1.2 get Syntax

You can query the current parameters of test resources using:

```
<module-index>/<port-index> <command> [<indices>] ?
```

The chassis responds with a line using identical syntax to the change-command, containing the current parameters. These responses can, therefore, be *''replayed"* back to the chassis to re-establish the parameter from a previous query.

This is actually the core of the load/save mechanism of the ValkyrieManager, as you can see by using an ordinary text editor to inspect the local files produced by saving. You can also change the content if you want to; it is not interpreted by the Manager applications.

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

Examples:

```
C_OWNER ?
0 M_COMMENT ?
0 P_RESERVEDBY ?
0/5 PS_RATEPPS [3] ?
```

2.2 Concurrent Sessions

2.2.1 Manage CLI Session

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

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 *Python*, *Java*, *Tcl*, *Perl*, and *Telnet*. You may use client-side functionality to execute script commands conditionally or repetitively.

Important: All lines sent to the chassis must be terminated by `CR/LF`. You will need to include these characters explicitly if the connection library of your platform does not do so automatically.

Note: Also please make sure to read all the responses sent back to you from the chassis, so that the buffer is empty when the connection is eventually closed down.

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

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. The timeout interval can be changed with the `C_TIMEOUT` command so that for instance `C_TIMEOUT 99999` effectively disables the timeout.

2.2.2 Multiple Concurrent Sessions

The chassis support multiple concurrent scripting sessions, just like they support multiple concurrent interactive ValkyrieManager 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.

Multiple users can operate on the chassis simultaneously. To ensure smooth operation, access restrictions apply.

All parameters can be read by anyone as long as that user has logged on using the `C_LOGON` command. In order to set parameters on a chassis, module, or port, the corresponding resource must be reserved by that user. Users are identified by name, which is configured using the `C_OWNER` command. Reservation state query and reservations can be done using the `C/M/P_RESERVATION` and `C/M/P_RESERVEDBY` commands.

Example:

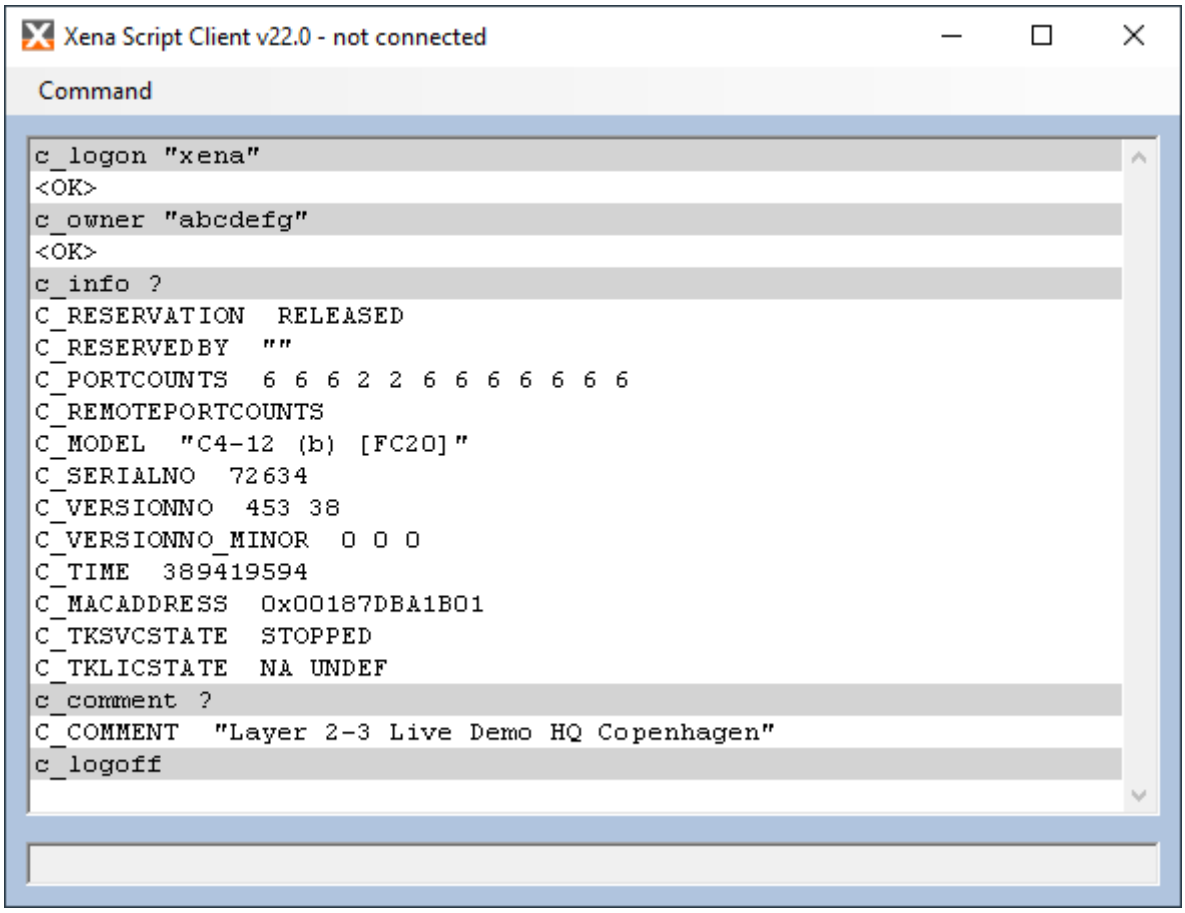


Fig. 2.1: XenaScriptClient

Logging on to a chassis and reserving port 0 and 1 is done by the following commands:

```
C_LOGON "xena"
C_OWNER "JohnDoe"

1/0 P_RESERVATION reserve
1/1 P_RESERVATION reserve
```

Resources reserved by a user can be released by the same user using the command:

```
1/0 P_RESERVATION release
```

Resources can also be released by other users using the command:

```
1/0 P_RESERVATION relinquish
```

A description of what a resource is used for can be given using `C/P_COMMENT`.

Example:

```
C_COMMENT "Live demo chassis. Resources can be relinquished without␣
↪warning"
```

or

```
1/0 P_COMMENT "Port used until 5pm sat oct. 5, after which it can be␣
↪relinquished"
1/1 P_COMMENT "Port used until 5pm sat oct. 5, after which it can be␣
↪relinquished"
```

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

```
;This this a comment
```

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

2.3 Parameter Data Types

A parameter is specified using one of the following formats:

- `byte` (`B`): 8-bit decimal integer, range [0 to 255].
- `integer` (`I`): 32-bit decimal integer, range [-2,147,483,648 to 2,147,483,647].
- `long` (`L`): 64-bit decimal integer, range [-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807].
- `hex` (`H`): two hexadecimal digits prefixed by `0x`, e.g. 0xF7. Some parameters consist of multiples of `hex`, for example 0x1234. They are denoted as `hex<n>` or `H..H`, where <n> is the number of `H`. For example, MAC address is of type `hex6` (`HHHHHH`), and IPv6 address is of type `hex16` (`HHHHHHHHHHHHHHHH`).

- `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 the reservation.
- `address` (`A`): a dot-separated IPv4 address, e.g. 192.168.1.200.
- `integer list` (`I*`): Some commands allow a variable number of parameters of `integer`, and they are written with spaces in between.
- `byte list` (`B*`): Some commands allow a variable number of parameters of `byte`, and they are written with spaces in between.
- `hex list` (`H*`): Some commands allow a variable number of parameters of `hex`, and multiple bytes can be specified using a single `0x` prefix, e.g. 0xF700ABCD2233 , **without spaces**.
- `address list` (`A*`): Some commands allow a variable number of parameters of `address`, and they are written with spaces in between.

Note: `hex<n>` (`H.....H`) is different from `hex list` (`H*`). The length of a parameter of type `hex<n>` (`H.....H`) is determined, while the length of a parameter of type `hex list` (`H*`) is dynamically dependent on the current status.

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

2.4 Status Messages

The `set` 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:

- `<NOCONNECTIONS>` Chassis has no available connection slots.
- `<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.
- `<NOTREADABLE>` The command is write-only.
- `<NOTWRITABLE>` The command is read-only.
- `<NOTVALID>` The operation is not valid in the current chassis state, e.g. because traffic is on.
- `<BADPARAMETER>` Invalid CLI command.
- `<BADMODULE>` The module index value is out of bounds.

- <BADPORT> The port index value is out of bounds.
- <BADINDEX> A command sub-index value is wrong.
- <BADSIZE> The size of a parameter is invalid.
- <BADVALUE> A parameter is invalid.
- <FAILED> An operation failed to produce a result.
- <NOTSUPPORTED> Feature not supported.
- <MEMORYFAILURE> Failed to allocate memory.
- <PENDING> Status return will wait until command is executed.
- <MODULE_OPERATION_NOT_SUPPORTED_BY_CHASSIS> Module is not supported by chassis - e.g. because multi-image requires x64 OS..
- <XLSFAILED> Could not establish connection to Xena License Server.
- <XLSDENIED> Request for resource rejected by Xena License Server.
- <XLSINVALID> Request for wrong resource type.

In case of a plain syntax error, misspelled command name, or 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
```

2.5 Defaults and Wildcarding

Note: Defaults and wildcarding is only supported by the XenaScriptClient application, and partially by ValkyrieManager's integrated Script Client.

The XOA CLI environment provides you with optional default values for the module index and port index, allowing you to set and get without providing the module and port index explicitly.

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

- <module-index>/<port-index>, set default module and port to the specified values.
- <port-index>, set default port to the specified value, retaining the default module.
- -/-, disable the default module and port.
- -, disable the default port, retaining the default module.
- <module-index>/-, set the default module and disable the default port.
- ?, show the current default module and port.

When a default module and port is provided, commands 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 wildcarding 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.

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

```
0/* P_INTERFRAMEGAP ?
```

Wildcards cannot be used as default values, but the default and wildcard mechanisms can be combined, for instance, to use a default module together with a wildcarded 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.

Important: Wildcard * is not supported by ValkyrieManager's integrated Script Client.

2.6 Special Commands

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

- `SYNC`. This command simply produces a reply of, which can be helpful when parsing and delimiting the lines returned from the chassis, in particular when using multi-command 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 `<n>` number of seconds, up to 60, and then produces a reply of. 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 XenaScriptingClient.
- `HELP ''<command>''`. Gives you a brief overview of the required indices and parameters for `''<command>''`. You are allowed to specify only a prefix of the command name, which will then give you the overview for each matching command, e.g. `HELP ''P_''` for all port-level commands. The summary of the required parameters 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 parameter where the two relevant parameters can be specified using coded names.

2.7 Relation to ValkyrieManager

What you can do with ValkyrieManager application can also be done using XOA CLI commands, and the correspondence is quite straightforward. *For example*, just as ValkyrieManager's **Port Properties** panel has a field for viewing and changing a port's minimum inter-frame gap, the XOA CLI interface can view and change the `P_INTERFRAMEGAP` command for doing the same. The same applies to most of the other fields of ValkyrieManager's *GUI*.

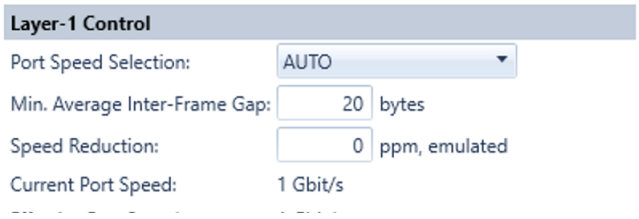


Fig. 2.2: Min. Average Inter-Frame Gap field in ValkyrieManager

However, there are a areas where ValkyrieManager has more advanced functionality, which is missing in the CLI commands. 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, ValkyrieManager 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.** ValkyrieManager 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.** ValkyrieManager 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.** ValkyrieManager 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, ValkyrieManager 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 command 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, ValkyrieManager 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 XOA CLI environment provides *Defaults and Wildcarding* across modules and ports, which is not available through ValkyrieManager.

2.8 Relation to Port Configuration Files

ValkyrieManager saves test port configurations (.xpc file) in the exact same CLI command format as used by the XOA CLI. This makes it very easy to go back and forth between a ValkyrieManager environment and XOA CLI. For example, exporting a port configuration from ValkyrieCLIManager generates a configuration file in a simple text format that can be edited using a text editing tool such as Microsoft Notepad. It can then be imported back into ValkyrieManager.

The seamless interaction between port configuration files and the XOA CLI accelerates your scripting learning curve, letting you get more done quicker as complex test port configurations can easily be defined in ValkyrieManager, and then exported to a text based configuration file, which in turn can be cut & pasted into your scripting tool environment.

Here is an example of a port configuration file.

Listing 2.1: port_config.xpc

```
;XENAPORT
;FormatVersion: 2
;Savedby: ValkyrieManager (v1.80.8196.2)
;Testbed: Default testbed
;Chassis: New chassis S.454, D.40)
;ChassisSerial: xxxxxxxx
;ModuleModel: M6RJ45N[b]
;Port: 0/0
;ModuleSerial: xxxxxx
;ModuleVersion: 312
;Global: S+C+T+R+
P_RESET
P_SPEEDSELECTION  AUTO
P_COMMENT  "Port number 0"
P_TXENABLE  ON
P_INTERFRAMEGAP  20
P_FLASH  OFF
P_AUTONEGSELECTION  ON
P_SPEEDREDUCTION  -1
P_MACADDRESS  0x04F4BCA53E60
P_IPADDRESS  0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0
P_MULTICAST  0.0.0.0 OFF 25
P_MULTICASTEXT  0.0.0.0 OFF 25 IGMPV2
P_MULTICASTHDR  0 NOHDR 0 0 DEI_OFF
P_MCSRCLIST  0.0.0.0
P_ARPREPLY  OFF
P_PINGREPLY  OFF
P_IPV6ADDRESS  0x00000000000000000000000000000000␣
↪0x00000000000000000000000000000000 128 128
P_ARPV6REPLY  OFF
P_PINGV6REPLY  OFF
P_ARPRXTABLE
P_NDPRXTABLE
P_PAUSE  OFF
P_PFCENABLE  OFF OFF OFF OFF OFF OFF OFF OFF
P_RANDOMSEED  0
P_LATENCYOFFSET  0
P_LATENCYMODE  LAST2LAST
P_TXTIMELIMIT  0
P_TXBURSTPERIOD  0
P_TXPACKETLIMIT  0
P_TXMODE  NORMAL
P_MAXHEADERLENGTH  128
P_AUTOTRAIN  0
P_LOOPBACK  NONE
```

(continues on next page)

(continued from previous page)

```
P_CHECKSUM  OFF
P_GAPMONITOR  0 0
P_MIXWEIGHTS  0 0 0 0 57 3 5 1 2 5 1 4 4 18 0 0
P_TXDELAY  0
P_TPLDMODE  NORMAL
P_DYNAMIC  OFF
P_PAYLOADMODE  NORMAL
P_LPENABLE  OFF
P_LPTXMODE  OFF
PEC_INDICES
PS_INDICES  0
PS_ENABLE  [0]  ON
PS_PACKETLIMIT  [0]  -1
PS_COMMENT  [0]  "Stream number 0"
PS_RATEFRACTION  [0]  500000
PS_BURST  [0]  -1 100
PS_BURSTGAP  [0]  0 0
PS_HEADERPROTOCOL  [0]  ETHERNET
PS_PACKETHEADER  [0]  0x00000000000004F4BCA53E60FFFF
PS_MODIFIERCOUNT  [0]  0
PS_PACKETLENGTH  [0]  FIXED 64 1518
PS_PAYLOAD  [0]  INCREMENTING 0x00
PS_TPLDID  [0]  0
PS_INSERTFCS  [0]  ON
PS_IPV4GATEWAY  [0]  0.0.0.0
PS_IPV6GATEWAY  [0]  0x00000000000000000000000000000000
PS_PFCPRIORITY  [0]  VLAN_PCP
PM_INDICES
PL_INDICES
PF_INDICES
PC_TRIGGER  ON 0 FULL 0
PC_KEEP  ALL 0 -1
PD_INDICES
```

2.9 Sample XOA CLI Script

2.9.1 Input

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

Listing 2.2: sample_input.txt

```
; 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 command 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 commands 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
; commands 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 type of port this is by querying for the interface␣
↪type:
P_INTERFACE ?

; Now relinquish and reserve the port, clear any existing␣
↪configuration,
; and set it in loop-mode:
P_RESERVATION RELINQUISH
P_RESERVATION RESERVE
P_RESET
```

(continues on next page)

(continued from previous page)

```
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 and 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 commands a at time using some special
; pseudo-commands. Here we'll query for the complete stream definition.
; This will give us all the commands 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-command query you cannot
; immediately tell which command 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 commands coming:
SYNC

; We're finally ready to run some traffic, but before we start the␣
```

(continues on next page)

(continued from previous page)

```
↪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-command 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␣
```

(continues on next page)

(continued from previous page)

```
↪configuration.
```

2.9.2 Output

Below you can download a file containing 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 XenaScriptClient.

You need to do a line-by-line correlation of the two lists in order to fully understand the output.

Note: There are sections of blank lines in the output corresponding to the comment lines in the input.

Listing 2.3: sample_output.txt

```
;; 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 command 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 commands 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"
;
```

(continues on next page)

(continued from previous page)

```
;; We now set a default port for the session so that all port-specific
;; commands 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 type of port this is by querying for the interface␣
↪type:
;P_INTERFACE ?
;
;; Now relinquish and reserve the port, clear any existing␣
↪configuration,
;; and set it in loop-mode:
;P_RESERVATION RELINQUISH
;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 and 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:
```

(continues on next page)

(continued from previous page)

```
;PS_PACKETLENGTH [10] ?
;P_MACADDRESS ?
;; You can also ask for multiple commands a at time using some special
;; pseudo-commands. Here we'll query for the complete stream␣
↪definition.
;; This will give us all the commands 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-command query you cannot
;; immediately tell which command 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 commands 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
```

(continues on next page)

(continued from previous page)

```
;; byte. We'll use another multi-command 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.
```

```
<OK>
<OK>
```

(continues on next page)

(continued from previous page)

```
P_INTERFACE  "T1 100/1000M [Dual] [Auto]"



<NOTVALID>
<OK>
<OK>
<OK>









<OK>
<OK>
<OK>
<OK>
<OK>
<OK>
<OK>
<OK>
<OK>
<OK>




PS_PACKETLENGTH  [10]  RANDOM 100 200
P_MACADDRESS  0x04F4BC94DAE0




PS_ENABLE  [10]  ON
PS_PACKETLIMIT  [10]  1000
```

(continues on next page)

(continued from previous page)

```
PS_COMMENT  [10]  "Example stream of 1000 packets"
PS_RATEFRACTION  [10]  500000
PS_BURST  [10]  -1 100
PS_BURSTGAP  [10]  0 0
PS_HEADERPROTOCOL  [10]  ETHERNET
PS_PACKETHEADER  [10]  0x00000000000004F4BC94DAE0FFFF
PS_MODIFIERCOUNT  [10]  1
PS_MODIFIER  [10,0]  5 0xFF000000 DEC 1
PS_MODIFIERRANGE  [10,0]  0 1 65535
PS_PACKETLENGTH  [10]  RANDOM 100 200
PS_PAYLOAD  [10]  INCREMENTING
PS_TPLDID  [10]  77
PS_INSERTFCS  [10]  ON
PS_IPV4GATEWAY  [10]  0.0.0.0
PS_IPV6GATEWAY  [10]  0x00000000000000000000000000000000
PS_PFCPRIORITY  [10]  VLAN_PCP









<SYNC>









<OK>
<OK>
PC_STATS  0 1 380727300691184
PC_PACKET  [0]  0x001122334455AABBCCDDEEFF2222FEDCBA98765432
1000000000FD07070707070707070707070707




<OK>
<RESUME>
PT_TOTAL  0 0 149400 1001
PT_NOTPLD  0 0 26 1
PT_EXTRA  0 0 0 0 0 0 0 0 0 0 0
PT_STREAM  [10]  0 0 149374 1000
P_RECEIVESYNC  IN_SYNC
PR_TOTAL  0 0 149374 1000
```

(continues on next page)

(continued from previous page)

```
PR_NOTPLD  0 0 0 0
PR_EXTRA  1 0 0 0 0 0 0 0
PR_PFCSTATS  0 0 0 0 0 0 0 0 0
PR_TPLDS  77
PR_TPLDTRAFFIC  [77]  0 0 149374 1000
PR_TPLDERRORS  [77]  0 0 0 0
PR_TPLDLATENCY  [77]  53 53 53 53 53 53
PR_TPLDJITTER  [77]  -1 -1 -1 -1 -1 -1




PC_STATS  1 407 380727300691184
PC_EXTRA  [1]  380727384593744 0 9527447 142
PC_PACKET  [1] ␣
↪0x0000000000FF04F4BC94DAE0FFFF0E0F101112131415161718191A1B1C1D1E
1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D
3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C
5D5E5F606162636465666768696A6B6C6D6E6F707172737475000000A01072
3D004D0E8000025DA67CAFBD79A06975CFA6
PC_EXTRA  [2]  380727384596744 0 190 187
PC_PACKET  [2] ␣
↪0x0000000000FE04F4BC94DAE0FFFF0E0F101112131415161718191A1B1C1D1E
1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D
3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C
5D5E5F606162636465666768696A6B6C6D6E6F707172737475767778797A7B
7C7D7E7F808182838485868788898A8B8C8D8E8F909192939495969798999A
9B9C9D9E9FA0A1A2000001A01073B5004D0E0000335D667DEEB484A0E06076C2
PC_EXTRA  [3]  380727384599360 0 189 143
PC_PACKET  [3] ␣
↪0x0000000000FD04F4BC94DAE0FFFF0E0F101112131415161718191A1B1C1D1E
1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D
3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C
5D5E5F606162636465666768696A6B6C6D6E6F707172737475760000002A010
74FD004D0E0000575DB67A6EFF28A0090E50FA
PC_EXTRA  [4]  380727384602416 0 191 185
PC_PACKET  [4] ␣
↪0x0000000000FC04F4BC94DAE0FFFF0E0F101112131415161718191A1B1C1D1E
1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D
3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C
5D5E5F606162636465666768696A6B6C6D6E6F707172737475767778797A7B
7C7D7E7F808182838485868788898A8B8C8D8E8F909192939495969798999A
9B9C9D9E9FA0000003A010767C004D0E0000F75D6B78297FC6A0181B0A5C
PC_EXTRA  [5]  380727384604776 0 189 106
PC_PACKET  [5] ␣
```

(continues on next page)

(continued from previous page)

```
↪0x0000000000FB04F4BC94DAE0FFFF0E0F10111213141516171819 1A1B1C1D1E
1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D
3E3F404142434445464748494A4B4C4D4E4F5051000004A01077A4004D0E00
002C5DC3798EA0ADA0A2FEAF0D



<OK>
<OK>
```

CHAPTER

THREE

CLI REFERENCE

3.1 Chassis

Chassis commands that deal with basic information and configuration of the chassis itself (rather than its modules and test ports), as well as overall control of the scripting session. The chassis command names all have the form C_<xxx> and use neither a module index nor a port index.

3.1.1 Identification

C_COMMENT

code: 21

```
# set
C_COMMENT <comment>

# get
C_COMMENT ?
```

Description

The description of the chassis.

Actions

set, get

Parameters

1. `comment`: *string*, the description of the tester

Example

```
# set
input:  C_COMMENT "this a comment"
output: <OK>

# get
input:  C_COMMENT ?
output: C_COMMENT "this a comment"
```

C_MODEL

`code:  10`

```
# get
C_MODEL ?
```

Description

Gets the specific model of this Xena chassis.

Actions

get

Parameters

1. `model`: *string*, the model of the Xena tester

Example

```
# get
input:  C_MODEL ?
output: C_MODEL "C4-12 (b) [FC20]"
```

C_MODEL_NAME

code: 457

```
# get
C_MODEL_NAME ?
```

Description

Get the Xena chassis model name.

Actions

get

Parameters

1. name: *integer*, the model of the Xena tester
   - NA = 0
   - B720 = 1
   - B720D = 2
   - B2400 = 3
   - Z_01_T_C_ODIN = 4
   - Z_100_Q_C_LOKI = 5
   - Z_10_S_C_ODIN = 6
   - Z_10_C_C_ODIN = 7
   - Z_10_R_C_ODIN = 8
   - Z_10_S_X_C_ODIN = 9
   - Z_01_S_C_ODIN = 10
   - Z_01_S_X_C_ODIN = 11
   - Z_400_Q_C_THOR = 12
   - Z_400_Q_LE_C_THOR = 13
   - Z_800_Q_C_FREYA = 14
   - Z_800_O_C_FREYA = 15
   - Z_800_Q_A_C_FREYA = 16
   - Z_800_O_A_C_FREYA = 17

- E_100_Q_C_CHIMERA = 18

Example

```
# get
input:  C_MODEL_NAME ?
output: C_MODEL B2400
```

C_MODEL_NUMBER

code: 458

```
# get
C_MODEL_NUMBER ?
```

Description

Get the Xena chassis model number.

Actions

get

Parameters

1. model: *string*, the model of the Xena tester

Example

```
# get
input:  C_MODEL_NUMBER ?
output: C_MODEL_NUMBER "XB22"
```

C_NAME

code: 20

```
# set
C_NAME <chassis_name>

# get
C_NAME ?
```

Description

The name of the chassis, as it appears at various places in the 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.

Actions

set, get

Parameters

1. chassis_name: *string*, the name of the tester

Example

```
# set
input:  C_NAME "L23 Live Demo"
output: <OK>

# get
input:  C_NAME ?
output: C_NAME "L23 Live Demo"
```

C_OWNER

code: 2

```
# set
C_OWNER <username>

# get
C_OWNER ?
```

Description

Identify the owner of the management session. 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.

Maximum 32 ASCII characters.

Actions

set, get

Parameters

1. `username`: *owner*, the username of this chassis management session.

Example

```
# set
input:  C_OWNER "Bob"
output: <OK>

# get
input:  C_OWNER ?
output: C_OWNER "Bob"
```

C_PASSWORD

`code:  22`

```
# set
C_PASSWORD <password>

# get
C_PASSWORD ?
```

Description

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

Actions

set, get

Parameters

1. `password`: *string*, the password of the tester

Example

```
# set
input:  C_PASSWORD "abcd"
output: <OK>

# get
input:  C_PASSWORD ?
output: C_PASSWORD "abcd"
```

C_SERIALNO

`code:  11`

```
# get
C_SERIALNO ?
```

Description

Gets the unique serial number of this particular Xena chassis.

Actions

get

Parameters

1. `serial_number`: *integer*, the serial number of the Xena tester

Example

```
# get
input:  C_SERIALNO ?
output: C_SERIALNO 1
```

C_VERSIONNO

`code:  12`

```
# get
C_VERSIONNO ?
```

Description

Gets the major version numbers for the chassis firmware and the Xena PCI driver installed on the chassis.

Actions

get

Parameters

1. `chassis_major_version`: *integer*, the firmware major version number of the tester and the PCI driver version
2. `pci_driver_version`: *integer*, the firmware major version number of the tester and the PCI driver version

Example

```
# get
input:  C_VERSIONNO ?
output: C_VERSIONNO 423 2
```

C_VERSIONNO_MINOR

code: 56

```
# get
C_VERSIONNO_MINOR ?
```

Description

Gets the minor version number for the chassis firmware. The full version of the chassis firmware is thus where the number is obtained with the C_VERSIONNO command and the number is obtained with the C_VERSIONNO_MINOR command.

Actions

get

Parameters

1. chassis_minor_version: *integer*, the minor version number for the chassis firmware
2. reserved_1: *integer*, reserved, 0
3. reserved_2: *integer*, reserved, 0

Example

```
# get
input:  C_VERSIONNO_MINOR ?
output: C_VERSIONNO_MINOR 1 0 0
```

3.1.2 Management Address

C_DHCP

code: 25

```
# set
C_DHCP <on_off>

# get
C_DHCP ?
```

Description

Controls whether the chassis will use DHCP to get the management IP address.

Actions

set, get

Parameters

1. on_off: *byte*, whether DHCP is enabled or disabled.

- OFF = 0
- ON = 1

Example

```
# set
input:  C_DHCP OFF
output: <OK>

# get
input:  C_DHCP ?
output: C_DHCP OFF
```

C_HOSTNAME

code: 27

```
# set
C_HOSTNAME <hostname>

# get
C_HOSTNAME ?
```

Description

Get or set the chassis hostname used when DHCP is enabled.

Actions

set, get

Parameters

1. hostname: *string*, the chassis hostname

Example

```
# set
input:  C_HOSTNAME "xena-123456"
output: <OK>

# get
input:  C_HOSTNAME ?
output: C_HOSTNAME "xena-123456"
```

C_IPADDRESS

code: 24

```
# set
C_IPADDRESS <ipv4_address> <subnet_mask> <gateway>

# get
C_IPADDRESS ?
```

Description

The network configuration parameters of the chassis management port.

Actions

set, get

Parameters

1. `ipv4_address`: *address*, the static IP address of the chassis
2. `subnet_mask`: *address*, the subnet mask of the local network segment
3. `gateway`: *address*, the gateway of the local network segment

Example

```
# set
input:  C_IPADDRESS 192.168.1.100 255.255.255.0 192.168.1.1
output: <OK>

# get
input:  C_IPADDRESS ?
output: C_IPADDRESS 192.168.1.100 255.255.255.0 192.168.1.1
```

C_MACADDRESS

`code:  26`

```
# get
C_MACADDRESS ?
```

Description

Get the MAC address for the chassis management port.

Actions

get

Parameters

1. `mac_address`: *hex list*, the MAC address for the chassis management port

Example

```
# get
input:  C_MACADDRESS ?
output: C_MACADDRESS ?L
```

3.1.3 Control

C_DOWN

`code:  8`

```
# set
C_DOWN -1480937026 <operation>
```

Description

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

Actions

set

Parameters

1. `magic`: *integer*, must be the special value -1480937026.
2. `operation`: *byte*, what to do after shutting chassis down.

- `RESTART = 1`
- `POWEROFF = 2`

Example

```
# set
input:  C_DOWN -1480937026 RESTART
output: <OK>
```

C_FLASH

code: 28

```
# set
C_FLASH <on_off>

# get
C_FLASH ?
```

Description

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.

Actions

set, get

Parameters

1. on_off: *byte*, determines whether to blink all test port LEDs.
   - OFF = 0
   - ON = 1

Example

```
# set
input:  C_FLASH OFF
output: <OK>

# get
input:  C_FLASH ?
output: C_FLASH OFF
```

C_LOGOFF

code: 7

```
# set
C_LOGOFF
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  C_LOGOFF
output: <OK>
```

C_LOGON

code: 1

```
# set
C_LOGON <password>
```

Description

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

Actions

set

Parameters

1. `password`: *string*, password for creating a tester management session and logging on to the tester.

Example

```
# set
input:  C_LOGON '"xena"
output: <OK>
```

C_TRAFFIC

`code: 55`

```
# set
C_TRAFFIC <on_off> <module_ports>
```

Description

Starts or stops the traffic on a number of ports on the chassis simultaneously.
The ports are identified by pairs of integers (module port).

Actions

set

Parameters

1. `on_off`: *byte*, determines whether to start or stop traffic generation
   - `OFF = 0`
   - `ON = 1`
2. `module_ports`: *integer list*, specifies ports on modules, which should stop or start generating traffic

Example

```
# set
input:  C_TRAFFIC OFF 0 0 0 1
output: <OK>
```

C_TRAFFICSYNC

code: 70

```
# set
C_TRAFFICSYNC <on_off> <timestamp> <module_ports>

# get
C_TRAFFICSYNC ?
```

Description

Works just as the C_TRAFFIC command described above with an additional option to specify a point in time where traffic should be started. This can be used to start traffic simultaneously on multiple chassis. The ports are identified by pairs of integers (module port).

Note: This requires that the chassis in question all use the TimeKeeper option to keep their CPU clocks synchronized.

Actions

set, get

Parameters

1. on_off: *byte*, determines whether to start or stop traffic generation
   - OFF = 0
   - ON = 1
2. timestamp: *long integer*, the time where traffic should be started, expressed as the number of seconds since January 1 2010, 00
3. module_ports: *integer list*, specifies ports on modules, which should stop or start traffic generation.

Example

```
# set
input:  C_TRAFFICSYNC OFF 2147483647 0 0 0 1
output: <OK>

# get
input:  C_TRAFFICSYNC ?
output: C_TRAFFICSYNC OFF 2147483647 0 0 0 1
```

3.1.4 Management Session

C_INDICES

code: 40

```
# get
C_INDICES ?
```

Description

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

Actions

get

Parameters

1. `session_ids`: *integer list*, the session indices for all current sessions on the chassis

Example

```
# get
input:  C_INDICES ?
output: C_INDICES 0 1
```

C_KEEPALIVE

code: 3

```
# get
C_KEEPALIVE ?
```

Description

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

Actions

get

Parameters

1. `tick_count`: *integer*, an increasing number from the chassis.

Example

```
# get
input:  C_KEEPALIVE ?
output: C_KEEPALIVE 10
```

C_MULTIUSER

code: 62

```
# set
C_MULTIUSER <on_off>

# get
C_MULTIUSER ?
```

Description

Enable or disable the ability to control one resource from several different TCP connections of the same username (C_OWNER). By default, C_MULTIUSER is off, implying that if there are multiple TCP connections to the tester associated with the **same username**, only one connect can perform set action to a particular port/module/chassis at a time. Enabling C_MULTIUSER permits connections with the same username to simultaneously execute set operations on a port/module/chassis.

Actions

set, get

Parameters

1. on_off: *byte*, enable or disable the ability to control one resource from several different TCP connections

- OFF = 0
- ON = 1

Example

```
# set
input:  C_MULTIUSER ON
output: <OK>

# get
input:  C_MULTIUSER ?
output: C_MULTIUSER ON
```

C_STATSESSION

code: 41

```
# get
C_STATSESSION [<session_index>] ?
```

Description

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

Actions

get

Parameters

1. `session_type`: *byte*, type of session
   - `MANAGER = 1`
   - `SCRIPT = 2`
2. `ipv4_address`: *address*, client IP address
3. `owner`: *string*, the name of the session owner
4. `operation_count`: *long integer*, number of operations done during the session
5. `requested_byte_count`: *long integer*, number of bytes received by the chassis
6. `responded_byte_count`: *long integer*, number of bytes sent by the chassis

Example

```
# get
input:  C_STATSESSION [0] ?
output: C_STATSESSION [0] MANAGER 192.168.1.100 "Bob" 123456789123␣
↪123456789123 123456789123
```

C_TIMEOUT

code: 4

```
# set
C_TIMEOUT <second_count>

# get
C_TIMEOUT ?
```

Description

The maximum number of idle seconds allowed before the connection is timed out by the tester.

Actions

set, get

Parameters

1. `second_count`: *integer*, the maximum idle interval, default is 130 seconds.

Example

```
# set
input:  C_TIMEOUT 100
output: <OK>

# get
input:  C_TIMEOUT ?
output: C_TIMEOUT 100
```

3.1.5 Status

C_PORTCOUNTS

`code:  13`

```
# get
C_PORTCOUNTS ?
```

Description

Gets 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.

Actions

get

Parameters

1. `port_counts`: *short integer list*, the number of ports of each module slot of the tester, 0 for an empty slot.

Example

```
# get
input:  C_PORTCOUNTS ?
output: C_PORTCOUNTS 123 123
```

C_PORTERRORS

`code:  16`

```
# get
C_PORTERRORS ?
```

Description

Gets the number of errors detected across all streams on each port of each test module of the chassis. The counts are ordered in sequence with those of the module in the lowest numbered chassis slot first. Empty slots are skipped so that a chassis with a 6-port and a 2-port test module will return eight counts regardless of which slots they are in.

Note: CFP modules return eight error counts since they can be configured as up to eight 10G ports. When in 100G and 40G mode only the first one or two counts are significant.

Note: FCS errors are included, which leads to double-counting for streams detecting lost packets using the test payload mechanism.

Actions

get

Parameters

1. `error_count`: *long integer list*, the total number of errors across all streams, and including FCS errors.

Example

```
# get
input:  C_PORTERRORS ?
output: C_PORTERRORS 123456789123 123456789124
```

C_REMOTEPORTCOUNTS

`code:  17`

```
# get
C_REMOTEPORTCOUNTS ?
```

Description

Gets the number of ports of each remote module. A remote module is a relative to the xenaserver, for example, xenal47server. The first integer in the returned list is always 0 because it represents the xenaserver, which is not a remote module.

Actions

get

Parameters

1. `port_counts`: *short integer list*, the number of ports of each module slot of the tester, 0 for an empty slot.

Example

```
# get
input:  C_REMOTEPORTCOUNTS ?
output: C_REMOTEPORTCOUNTS 123 123
```

C_RESERVATION

code: 5

```
# set
C_RESERVATION <operation>

# get
C_RESERVATION ?
```

Description

You set this command 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 for other users.

Note: Before reserve Tester need to reserve all the ports on it, otherwise <STATUS_NOTVALID>

Actions

set, get

Parameters

1. operation: *byte*, reservation operation to be performed.

- RELEASE = 0
- RESERVE = 1
- RELINQUISH = 2

Example

```
# set
input:  C_RESERVATION RELEASE
output: <OK>

# get
input:  C_RESERVATION ?
output: C_RESERVATION RELEASE
```

C_RESERVEDBY

code: 6

```
# get
C_RESERVEDBY ?
```

Description

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

Actions

get

Parameters

1. `username`: *string*, the username of the current owner of the tester.

Example

```
# get
input:  C_RESERVEDBY ?
output: C_RESERVEDBY "Bob"
```

C_TEMPERATURE

code: 31

```
# get
C_TEMPERATURE ?
```

Description

Get chassis temperature readings, if supported. Unit is millidegree Celsius.

Actions

get

Parameters

1. `mb1_temperature`: *integer*, the temperature of motherboard 1 (millidegree Celsius)
2. `mb2_temperature`: *integer*, the temperature of motherboard 2 (millidegree Celsius)
3. `cpu_temperature`: *integer*, the temperature of CPU (millidegree Celsius)

Example

```
# get
input:  C_TEMPERATURE ?
output: C_TEMPERATURE 1 1 1
```

C_TIME

code: 69

```
# get
C_TIME ?
```

Description

Get local chassis time in seconds.

Actions

get

Parameters

1. `local_time`: *long integer*, local chassis time in seconds

Example

```
# get
input:  C_TIME ?
output: C_TIME 123456789123
```

3.1.6 Time Service

C_TKCONFIG

`code:  67`

```
# set
C_TKCONFIG <config_file>

# get
C_TKCONFIG ?
```

Description

TimeKeeper config file content.

Actions

set, get

Parameters

1. `config_file`: *string*, TimeKeeper config file content

Example

```
# set
input:  C_TKCONFIG "a string"
output: <OK>

# get
input:  C_TKCONFIG ?
output: C_TKCONFIG "a string"
```

C_TKGPSSTATE

`code:  68`

```
# get
C_TKGPSSTATE ?
```

Description

Get TimeKeeper GPS status.

Actions

get

Parameters

1. `status`: *string*, TimeKeeper GPS status

Example

```
# get
input:  C_TKGPSSTATE ?
output: C_TKGPSSTATE 0
```

C_TKLICFILE

code: 49

```
# set
C_TKLICFILE <license_content>

# get
C_TKLICFILE ?
```

Description

Get Xena TimeKeeper license file content.

Actions

set, get

Parameters

1. `license_content`: *short integer list*, Xena TimeKeeper license file content

Example

```
# set
input:  C_TKLICFILE "a string"
output: <OK>

# get
input:  C_TKLICFILE ?
output: C_TKLICFILE "a string"
```

C_TKLICSTATE

code: 50

```
# get
C_TKLICSTATE ?
```

Description

Get the state of the Xena TimeKeeper license file content.

Actions

get

Parameters

1. **lice1se_file_state:** *byte*, **timekeeper license state**
   - NA = 0
   - INV = 1
   - VALID = 2
2. **license_type:** *byte*, **license type**
   - UNDEF = 0
   - CLIENT = 1
   - SERVER = 2
3. **license_errors:** *integer list*, **license errors**
   - NO_LICENSE_ERROR = 0
   - INVALID_SERIALNO = 1
   - INVALID_CHASSISTYPE = 2

Example

```
# get
input:  C_TKLICSTATE ?
output: C_TKLICSTATE NA UNDEF NO_LICENSE_ERROR
```

C_TKSTATUS

code: 65

```
# get
C_TKSTATUS ?
```

Description

Report TimeKeeper version and status.

Actions

get

Parameters

1. `status_string`: *string*, Version, TimeKeeper license expiration, and TimeKeeper status. The string is formatted as shown in the example below.

Example

```
# get
input:  C_TKSTATUS ?
output: C_TKSTATUS  0 84 105 109 101 75 101 101 112 101 114 32 83 116␣
↪97 116 117 115 10 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61␣
↪61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61␣
↪61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61␣
↪61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 10 76 105 99 101␣
↪110 115 101 32 105 115 32 105 110 118 97 108 105 100 10 84 105 109␣
↪101 75 101 101 112 101 114 32 105 115 32 110 111 116 32 114 117 110␣
↪110 105 110 103 10 0
```

C_TKSTATUSEXT

code: 71

```
# get
C_TKSTATUSEXT ?
```

Description

Report TimeKeeper version and status (extended version).

Actions

get

Parameters

1. `status_string`: *string*, extended status in JSON format. The string is formatted as shown in the example below.

Example

```
# get
input:  C_TKSTATUSEXT ?
output: C_TKSTATUSEXT "a string"
```

C_TKSVCSTATE

`code:  66`

```
# set
C_TKSVCSTATE <state>

# get
C_TKSVCSTATE ?
```

Description

Get and control TimeKeeper service state.

Actions

set, get

Parameters

1. `state`: *byte*, TimeKeeper service state
   - `STOP = 0`
   - `START = 1`
   - `RESTART = 2`

Example

```
# set
input:  C_TKSVCSTATE STOP
output: <OK>

# get
input:  C_TKSVCSTATE ?
output: C_TKSVCSTATE STOP
```

3.1.7 Capability

C_CAPABILITIES

`code:  9`

```
# get
C_CAPABILITIES ?
```

Description

A series of integer values specifying various internal limits (aka. capabilities) of the chassis.

Actions

get

Parameters

1. `version`: *integer*, A series of integer values specifying various internal limits
2. `max_name_len`: *integer*, chassis software build number
3. `max_comment_len`: *integer*, max ASCII characters in chassis name
4. `max_password_len`: *integer*, max ASCII characters in chassis comment
5. `max_ext_rate`: *integer*, max ASCII characters in chassis password
6. `max_session_count`: *integer*, maximum rate for external traffic
7. `max_chain_depth`: *integer*, max number of management and scripting sessions
8. `max_module_count`: *integer*, max chain index
9. `max_protocol_count`: *integer*, maximum number of Valkyrie modules
10. `can_stream_based_arp`: *integer*, max protocol segments in a packet
11. `can_sync_traffic_start`: *integer*, does server support stream-based ARP/NDP?
12. `can_read_log_files`: *integer*, does server support synchronous traffic start?
13. `can_par_module_upgrade`: *integer*, can clients read debug log files from server?
14. `can_upgrade_timekeeper`: *integer*, can server handle parallel module upgrades?
15. `can_custom_defaults`: *integer*, is server capable of upgrading the TimeKeeper application?
16. `can_latency_f2f`: *integer*, can server handle custom default values for XMP parameters?
17. `max_owner_name_length`: *integer*, can server handle first-to-first latency mode?
18. `can_read_temperatures`: *integer*, max number of ASCII characters in `C_OWNER` name

Example

```
# get
input:  C_CAPABILITIES ?
output: C_CAPABILITIES 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

3.1.8 Misc

C_BUILDSTRING

code: 19

```
# get
C_BUILDSTRING ?
```

Description

Identify the hostname of the PC that builds the xenaserver. It uniquely identifies the build of a xenaserver.

Actions

get

Parameters

1. build_string: *string*, build string that identifies the hostname of the PC that builds the xenaserver

Example

```
# get
input:  C_BUILDSTRING ?
output: C_BUILDSTRING "2022-11-01-030223[localhost.localdomain]0e22fae"
```

C_DEBUGLOGS

code: 30

```
# get
C_DEBUGLOGS ?
```

Description

Allows to dump all the logs of a chassis.

Actions

get

Parameters

1. `message_length`: *integer*, length of the message and all the logs of the chassis
2. `data`: *hex list*, length of the message and all the logs of the chassis

Example

```
# get
input:  C_DEBUGLOGS ?
output: C_DEBUGLOGS 1 0x57
```

C_FILEDATA

`code:  52`

```
# set
C_FILEDATA <offset> <data_bytes>
```

Description

Uploads a fragment of a file to the chassis.

Actions

set

Parameters

1. `offset`: *integer*, the position within the file
2. `data_bytes`: *hex list*, the data content of a section of the file

Example

```
# set
input:  C_FILEDATA 1␣
↪0x1E2E3E4E5E1E2E3E4E5E1E2E3E4E5E1E2E3E4E5E1E2E3E4E5E
output: <OK>
```

C_FILEFINISH

code: 53

```
# set
C_FILEFINISH
```

Description

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

Actions

set

Parameters

1. `magic`: *integer*,

Example

```
# set
input:  C_FILEFINISH
output: <OK>
```

C_FILESTART

code: 51

```
# set
C_FILESTART <file_type> <size> <time> <mode> <checksum> <name>
```

Description

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

Actions

set

Parameters

1. file_type: *hex4*, the file type, should be 1
2. size: *hex4*, the number of bytes in the file
3. time: *hex4*, the Linux date+time of the file
4. mode: *hex4*, the Linux permissions of the file
5. checksum: *hex4*, the checksum of the file
6. name: *string*, the name and location of the file, as a full path

Example

```
# set
input:  C_FILESTART 0x01 0x01 0x01020304 0x444 0xedfe44324 "filename"
output: <OK>
```

C_RESTCONTROL

code: 34

```
# set
C_RESTCONTROL <operation>
```

Description

Controls REST API server. This command should be used with extra care as it can affect other users using the server.

Actions

set

Parameters

1. operation: *byte*, what to do with the REST API server
   - START = 0
   - STOP = 1
   - RESTART = 2

Example

```
# set
input:  C_RESTCONTROL START
output: <OK>
```

C_RESTENABLE

code: 33

```
# set
C_RESTENABLE <on_off>

# get
C_RESTENABLE ?
```

Description

Controls whether the chassis will run REST API server or not. The command takes affect only after chassis reset. To start/stop REST API server use C_RESTCONTROL command.

Actions

set, get

Parameters

1. on_off: *byte*, determines whether REST API server should be enabled or disabled
   - OFF = 0
   - ON = 1

Example

```
# set
input:  C_RESTENABLE OFF
output: <OK>

# get
input:  C_RESTENABLE ?
output: C_RESTENABLE OFF
```

C_RESTPORT

code: 32

```
# set
C_RESTPORT <tcp_port>

# get
C_RESTPORT ?
```

Description

The TCP port used by the REST API server.

Actions

set, get

Parameters

1. `tcp_port`: *integer*, the TCP port number (default 57911)

Example

```
# set
input:  C_RESTPORT 1
output: <OK>

# get
input:  C_RESTPORT ?
output: C_RESTPORT 1
```

C_RESTSTATUS

`code:  35`

```
# get
C_RESTSTATUS ?
```

Description

Gets the REST API server operation status - whether it is active (running) or not. To get the admin status (whether the server is enabled or disabled) use `C_RESTCONTROL` command.

Actions

get

Parameters

1. `status`: *byte*, the operation status of th REST API server

- `SERVICE_OFF = 0`
- `SERVICE_ON = 1`

Example

```
# get
input:  C_RESTSTATUS ?
output: C_RESTSTATUS SERVICE_OFF
```

C_SCRIPT

`code:  64`

```
# set
C_SCRIPT <command_string>
```

Description

To load and save CLI commands e.g. port configuration, through the binary XMP session.

Actions

set

Parameters

1. `command_string`: *string*, text CLI command

Example

```
# set
input:  C_SCRIPT "C_MODEL ?"
output: <OK>
```

C_WATCHDOG

code: 36

```
# set
C_WATCHDOG <timer_value>

# get
C_WATCHDOG ?
```

Description

If the chassis stalls for a long time, when the timer expires the chassis will be rebooted automatically.

Actions

set, get

Parameters

1. `timer_value`: *integer*, the timer value that reboots the chassis

Example

```
# set
input:  C_WATCHDOG 1
output: <OK>

# get
input:  C_WATCHDOG ?
output: C_WATCHDOG 1
```

3.1.9 Full Config

C_CONFIG

code: 29

```
# get
C_CONFIG ?
```

Description

Return all setable chassis values

Actions

get

Parameters

Example

```
# get
input: C_CONFIG ?
```

3.2 Module

Module commands that deal with basic information about, and configuration of the test modules. The module command names all have the form `M_<xxx>` and require a module index id.

3.2.1 Identification

M_COMMENT

code: 86

```
# set
<module-index> M_COMMENT <comment>

# get
<module-index> M_COMMENT ?
```

Description

Gets the user-defined description string of a module.

Actions

set, get

Parameters

1. `comment`: *string*, the user-specified comment/description for the module

Example

```
# set
input:  0 M_COMMENT "comment"
output: <OK>

# get
input:  0 M_COMMENT ?
output: 0 M_COMMENT "comment"
```

M_MODEL

`code:  75`

```
# get
<module-index> M_MODEL ?
```

Description

Get the technical code of the module.

Actions

get

Parameters

1. `model` *string*, the legacy model P/N name of a Xena test module

Example

```
# get
input:  0 M_MODEL ?
output: 0 M_MODEL "M6SFP"
```

M_MODEL_NAME

`code:  459`

```
# get
<module-index> M_MODEL_NAME ?
```

Description

Get the model name of the module.

Actions

get

Parameters

1. `name` *integer*, model name of Xena module.
   - `NA = 0`
   - `Z_01_T_ODIN = 1`
   - `Z_100_Q_LOKI = 2`
   - `Z_10_S_ODIN = 3`
   - `Z_10_R_ODIN = 4`
   - `Z_10_S_X_ODIN = 5`
   - `Z_01_S_ODIN = 6`
   - `Z_01_S_X_ODIN = 7`
   - `Z_400_Q_THOR = 8`

- Z_400_Q_LE_THOR = 9
- Z_800_Q_FREYA = 10
- Z_800_O_FREYA = 11
- E_100_Q_CHIMERA = 12

Example

```
# get
input:  0 M_MODEL_NAME ?
output: 0 M_MODEL_NAME Z_800_O_FREYA
```

M_NAME

code: 99

```
# get
<module-index> M_NAME ?
```

Description

Gets the name of a module.

Actions

get

Parameters

1. name: *string*, the name of the module

Example

```
# get
input:  0 M_NAME ?
output: 0 M_NAME "module name"
```

M_REVISION

code: 100

```
# get
<module-index> M_REVISION ?
```

Description

Get the product code of the module.

Actions

get

Parameters

1. `revision`: *string*, the model P/N name of a Xena test module.

Example

```
# get
input:  0 M_REVISION ?
output: 0 M_REVISION "Freya-800G-1S-1P [a]"
```

M_SERIALNO

code: 76

```
# get
<module-index> M_SERIALNO ?
```

Description

Gets the unique serial number of a module.

Actions

get

Parameters

1. `serial_number`: *integer*, the serial number of this test module

Example

```
# get
input:  0 M_SERIALNO ?
output: 0 M_SERIALNO 1
```

M_VERSIONNO

`code:  77`

```
# get
<module-index> M_VERSIONNO ?
```

Description

Gets the version number of the hardware image installed on a module.

Actions

get

Parameters

1. `version`: *integer*, the hardware image version number of the test module

Example

```
# get
input:  0 M_VERSIONNO ?
output: 0 M_VERSIONNO 1
```

3.2.2 Media Configuration

M_CFPCONFIG

code: 85

Deprecated since version 1.3: replaced by M_CFPCONFIGEXT

```
# set
<module-index> M_CFPCONFIG <port_count> <port_speed>

# get
<module-index> M_CFPCONFIG ?
```

Description

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.

Actions

set, get

Parameters

1. port_count: *short integer*, number of ports
2. port_speed: *short integer*, port speed, in Gbps

Example

```
# set
input:  0 M_CFPCONFIG 4 25
output: <OK>

# get
input:  0 M_CFPCONFIG ?
output: 0 M_CFPCONFIG 4 25
```

M_CFPCONFIGEXT

code: 93

```
# set
<module-index> M_CFPCONFIGEXT <port_count> <port_speed_list>

# get
<module-index> M_CFPCONFIGEXT ?
```

Description

This property defines the current number of ports and the speed of each of them on 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: 2x10G, 4x10G, 8x10G, 2x25G, 4x25G, 8x25G, 1x40G, 2x40G, 2x50G, 4x50G, 8x50G, 1x100G, 2x100G, 4x100G, 2x200G, and 1x400G. (replaces M_CFPCONFIG)

Actions

set, get

Parameters

1. port_count: *integer*, port count
2. port_speed_list: *integer list*, corresponding speeds supported by the current module config in Mbps, length of the list equals to the value of <port_count>

Example

```
# set
input:  0 M_CFPCONFIGEXT 4 25000 25000 25000 25000
output: <OK>

# get
input:  0 M_CFPCONFIGEXT ?
output: 0 M_CFPCONFIGEXT 4 25000 25000 25000 25000
```

M_MEDIA

code: 342

```
# set
<module-index> M_MEDIA <media_type>

# get
<module-index> M_MEDIA ?
```

Description

For the test modules that support media configuration (check M_CAPABILITIES), this command sets the desired media type (front port).

Actions

set, get

Parameters

1. `media_type`: *byte*, the media type of the test module

- `CFP4 = 0`
- `QSFP28 = 1`
- `CXP = 2`
- `SFP28 = 3`
- `QSFP56 = 4`
- `QSFP_DD = 5`
- `SFP56 = 6`

- SFP_DD = 7
- SFP112 = 8
- QSFP_DD_NRZ = 9
- QSFP28_PAM4 = 10
- CFP = 99
- BASE_T1 = 100
- BASE_T1S = 101
- QSFPDD800 = 110
- QSFP112 = 111
- OSFP800 = 112
- QSFPDD800_ANLT = 113
- QSFP112_ANLT = 114
- OSFP800_ANLT = 115

Example

```
# set
input:  0 M_MEDIA CFP4
output: <OK>

# get
input:  0 M_MEDIA ?
output: 0 M_MEDIA CFP4
```

M_MEDIASUPPORT

code: 90

```
# get
<module-index> M_MEDIASUPPORT ?
```

Description

This command shows the available speeds on a module. The structure of the returned value is [<cage_type> <available_speed_count> [<ports_per_speed> <speed>]]. [<ports_per_speed> <speed>] are repeated until all speeds supported by the <cage_type> has been listed. [<cage_type> <available_speed_count>] are repeated for all cage types on the module including the related <ports_per_speed> <speed> information.

Actions

get

Parameters

1. `media_info_list`: *integer list*, a list of integers. The structure of the returned value is [<cage_type> <available_speed_count>[<ports_per_speed> <speed>]]. [<ports_per_speed> <speed>] are repeated until all speeds supported by the <cage_type> has been listed. [<cage_type> <available_speed_count>] are repeated for all cage types on the module including the related <ports_per_speed> <speed> information.

Example

```
# get
input:  0 M_MEDIASUPPORT ?
output: 0 M_MEDIASUPPORT 0 1
```

3.2.3 Status

M_HEALTH

`code:  456`

```
# get
<module-index> M_HEALTH ?
```

Description

Gets the module health information.

Actions

get

Parameters

1. `info`: *string*, Module health information json string

The JSON schema:

```
{
        "0":{
                "name": "Module",
                "data": {
                        "model": "MFREYA-800G-4S-1P",
                        "name": "Freya-800G-4S-1P",
                        "serial_number": 846275,
                        "version_number": 45680
                }
        },
        "1":{
                "name": "Cage",
                "data": [
                        {"insert_count": 9},
                        {"insert_count": 0}
                ]
        }
}
```

Example

```
# get
input:  0 M_HEALTH ?
output: 0 M_HEALTH  "{",34,"0",34,":{",34,"name",34,": ",34,"Module",
↪34,",",34,"data",34,": {",34,"model",34,": ",34,"MFREYA-800G-1S-1P",
↪34,",",34,"name",34,": ",34,"Freya-800G-1S-1P",34,",",34,"serial_
↪number",34,": 752973,",34,"version_number",34,": 52711}},",34,"1",34,
↪":{",34,"name",34,": ",34,"Cage",34,",",34,"data",34,": [{",34,
↪"insert_count",34,": 15}]}}"
```

M_PORTCOUNT

code: 80

```
# get
<module-index> M_PORTCOUNT ?
```

Description

Gets the maximum number of ports on a module.

Note: For a CFP-type module this number refers to the maximum number of ports possible on the module regardless of the media configuration. So if a CFP-type module can be set in for instance either 1x100G mode or 8x10G mode then this command will always return 8. If you want the current number of ports for a CFP-type module you need to read the *M_CFPCONFIG* command which returns the number of current ports.

Actions

get

Parameters

1. `port_count`: *integer*, the maximum number of ports on the test module

Example

```
# get
input:  0 M_PORTCOUNT ?
output: 0 M_PORTCOUNT 1
```

M_RESERVATION

code: 72

```
# set
<module-index> M_RESERVATION <operation>

# get
<module-index> M_RESERVATION ?
```

Description

Set this command 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 for other users.

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

Actions

set, get

Parameters

1. `operation`: *byte*, reservation operation to perform

- `RELEASE = 0`
- `RESERVE = 1`
- `RELINQUISH = 2`

Example

```
# set
input:  0 M_RESERVATION RELEASE
output: <OK>

# get
input:  0 M_RESERVATION ?
output: 0 M_RESERVATION RELEASE
```

M_RESERVEDBY

`code:  73`

```
# get
<module-index> M_RESERVEDBY ?
```

Description

Identify the user who has a module reserved. Returns an empty string if the module is not currently reserved by anyone.

Actions

get

Parameters

1. `username`: *string*, the username who has reserved the test module

Example

```
# get
input:  0 M_RESERVEDBY ?
output: 0 M_RESERVEDBY "peter"
```

M_STATUS

`code:  79`

```
# get
<module-index> M_STATUS ?
```

Description

Get status readings for the test module itself.

Actions

get

Parameters

1. `temperature`: *integer*, temperature of the main hardware chip, in degrees Celsius

Example

```
# get
input:  0 M_STATUS ?
output: 0 M_STATUS 1
```

3.2.4 Timing

M_TIMEADJUSTMENT

`code:  88`

```
# set
<module-index> M_TIMEADJUSTMENT <adjust>

# get
<module-index> M_TIMEADJUSTMENT ?
```

Description

Control time adjustment for module wall clock.

Actions

set, get

Parameters

1. `adjust`: *integer*, the time adjustment value for the module clock

Example

```
# set
input:  0 M_TIMEADJUSTMENT 1
output: <OK>

# get
input:  0 M_TIMEADJUSTMENT ?
output: 0 M_TIMEADJUSTMENT 1
```

M_TIMESYNC

code: 83

```
# set
<module-index> M_TIMESYNC <source>

# get
<module-index> M_TIMESYNC ?
```

Description

Control how the test module timestamp clock is running, either freely in the chassis or locked to an external system time. Running with free chassis time allows nano-second precision measurements of latencies, but only when the transmitting and receiving ports are in the same chassis. Running with locked external time enables inter-chassis latency measurements, but can introduce small time discontinuities as the test module time is adjusted.

Actions

set, get

Parameters

1. `source`: *byte*, the timing source of the test module timestamp clock

- `CHASSIS = 0`
- `EXTERNAL = 1`
- `MODULE = 2`

Example

```
# set
input:  0 M_TIMESYNC CHASSIS
output: <OK>

# get
input:  0 M_TIMESYNC ?
output: 0 M_TIMESYNC CHASSIS
```

M_TXCLOCKFILTER_NEW

code: 412

```
# set
<module-index> M_TXCLOCKFILTER_NEW <filter_bandwidth>

# get
<module-index> M_TXCLOCKFILTER_NEW ?
```

Description

For test modules with advanced timing features, the loop bandwidth on the TX clock filter.

Actions

set, get

Parameters

1. `filter_bandwidth`: *byte*, the setting of the loop bandwidth on the TX clock filter

- `BW103HZ = 1`
- `BW207HZ = 2`
- `BW416HZ = 3`
- `BW1683HZ = 4`
- `BW7019HZ = 5`

Example

```
# set
input:  0 M_TXCLOCKFILTER_NEW BW103HZ
output: <OK>

# get
input:  0 M_TXCLOCKFILTER_NEW ?
output: 0 M_TXCLOCKFILTER_NEW BW103HZ
```

M_TXCLOCKSOURCE_NEW

code: 410

```
# set
<module-index> M_TXCLOCKSOURCE_NEW <tx_clock>

# get
<module-index> M_TXCLOCKSOURCE_NEW ?
```

Description

For test modules with advanced timing features, select what clock drives the port TX rates.

Actions

set, get

Parameters

1. `tx_clock`: *byte*, the test module's TX clock source settings

- `MODULELOCALCLOCK = 0`
- `SMAINPUT = 1`
- `P0RXCLK = 2`
- `P1RXCLK = 3`
- `P2RXCLK = 4`
- `P3RXCLK = 5`
- `P4RXCLK = 6`
- `P5RXCLK = 7`

- P6RXCLK = 8
- P7RXCLK = 9

Example

```
# set
input:  0 M_TXCLOCKSOURCE_NEW MODULELOCALCLOCK
output: <OK>

# get
input:  0 M_TXCLOCKSOURCE_NEW ?
output: 0 M_TXCLOCKSOURCE_NEW MODULELOCALCLOCK
```

M_TXCLOCKSTATUS_NEW

code: 411

```
# get
<module-index> M_TXCLOCKSTATUS_NEW ?
```

Description

For test modules with advanced timing features, check whether a valid clock is present.

Actions

get

Parameters

1. status: *byte*, the status of whether a valid clock is present for the test module.

- OK = 0
- NOVALIDTXCLK = 1

Example

```
# get
input:  0 M_TXCLOCKSTATUS_NEW ?
output: 0 M_TXCLOCKSTATUS_NEW OK
```

3.2.5 Clock Sweep

M_CLOCKPPB

code: 94

```
# set
<module-index> M_CLOCKPPB <ppb>

# get
<module-index> M_CLOCKPPB ?
```

Description

Makes small adjustments to the local clock of the test module, which drives the TX rate of the test ports.

Actions

set, get

Parameters

1. ppb: *integer*, adjustment from nominal value, in parts-per-billion, positive or negative

Example

```
# set
input:  0 M_CLOCKPPB 1
output: <OK>

# get
input:  0 M_CLOCKPPB ?
output: 0 M_CLOCKPPB 1
```

M_CLOCKPPBSWEEP

code: 413

```
# set
<module-index> M_CLOCKPPBSWEEP <mode> <ppb_step> <step_delay> <max_ppb>
↪ <loops>

# get
<module-index> M_CLOCKPPBSWEEP ?
```

Description

Start and stop deviation sweep the local clock of the test module, which drives the TX rate of the test ports

Actions

set, get

Parameters

1. `mode`: *byte*: specifying the sweeping function.
   - `NONE`
   - `TRIANGLE`
2. `ppb_step`: *integer*, >=`0`, the numeric clock adjustment in ppb per step of the sweep. If set to 0, the sweep will use as small steps as possible, creating a "linear" sweep of the clock rate.
3. `step_delay`: *integer*, >`0`, the delay in µs between each step in the sweep. If ppb_step is 0: The total time in µs to sweep linearly from 0 to max_ppb.
4. `max_ppb`: *integer*, `!=0`, the numeric maximum clock adjustment. The sign of max_ppb determines if the sweep will start with positive or negative offsets. When the next step would exceed the limit set by max_ppb, the sweep changes direction, i.e. the deviation will sweep from 0 to `max_ppb`, to (`-max_ppb`), and back to 0.
5. `loops`: *integer*, >=`0`, the number of full sweeps performed. 0 means "indefinitely".

Example

```
# set
input:  0 M_CLOCKPPBSWEEP TRIANGLE 10000 1000000 100000 0
output: <OK>

# get
input:  0 M_CLOCKPPBSWEEP ?
output: 0 M_CLOCKPPBSWEEP TRIANGLE 10000 1000000 100000 0
```

Note: The sweep is independent of the M_CLOCKPPB parameter. I.e. the sweep uses the deviation set by M_CLOCKPPB as its zero point.

M_CLOCKSWEEPSTATUS

code: 414

```
# get
<module-index> M_CLOCKSWEEPSTATUS ?
```

Description

Return the current status of the M_CLOCKPPBSWEEP function.

Actions

get

Parameters

1. state: *byte*: specifying if a sweep is active.
   - OFF
   - SWEEPING
2. curr_sweep: *integer*, >=0, the current full sweep number, counting from 0.
3. curr_step: *integer*, >=0, the current step number inside the sweep, counting from 0.
4. max_steps: *integer*, >0, the total number of steps comprising a full sweep. For "linear" sweeps (ppb_step=0, see *M_CLOCKPPBSWEEP*) this number is determined by the chassis. In other cases, the number is implicitly given by the *M_CLOCKPPBSWEEP* parameters.

Example

```
# get
input:  0 M_CLOCKSWEEPSTATUS ?
output: 0 M_CLOCKSWEEPSTATUS SWEEPING 2 13 40
```

M_CLOCKSYNCSTATUS

code: 370

```
# get
<module-index> M_CLOCKSYNCSTATUS ?
```

Description

Get module's clock sync status.

Actions

get

Parameters

1. `m_clock_diff`: *long integer*, the test module's clock sync status
2. `m_correction`: *long integer*, the test module's clock sync status
3. `m_tune_is_increase`: *long integer*, the test module's clock sync status
4. `m_tune_value`: *long integer*, the test module's clock sync status
5. `m_is_steady_state`: *long integer*, the test module's clock sync status

Example

```
# get
input:  0 M_CLOCKSYNCSTATUS ?
output: 0 M_CLOCKSYNCSTATUS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

3.2.6 SMA

M_SMAINPUT

code: 95

```
# set
<module-index> M_SMAINPUT <sma_in>

# get
<module-index> M_SMAINPUT ?
```

Description

For test modules with SMA (SubMiniature version A) connectors, selects the function of the SMA input.

Actions

set, get

Parameters

1. sma_in: *byte*, the function of the SMA (SubMiniature version A) input of the module
   - NOTUSED = 0
   - TX2MHZ = 1
   - TX10MHZ = 2

Example

```
# set
input:  0 M_SMAINPUT NOTUSED
output: <OK>

# get
input:  0 M_SMAINPUT ?
output: 0 M_SMAINPUT NOTUSED
```

M_SMAOUTPUT

code: 96

```
# set
<module-index> M_SMAOUTPUT <sma_out>

# get
<module-index> M_SMAOUTPUT ?
```

Description

For test modules with SMA (SubMiniature version A) connectors, selects the function of the SMA output.

Actions

set, get

Parameters

1. sma_out: *byte*, the function of the SMA (SubMiniature version A) output of the module

- DISABLED = 0
- PASSTHROUGH = 1
- P0SOF = 2
- P1SOF = 3
- REF2MHZ = 4
- REF10MHZ = 5
- REF125MHZ = 6
- REF156MHZ = 7
- P0RXCLK = 8
- P1RXCLK = 9
- P0RXCLK2MHZ = 10
- P1RXCLK2MHZ = 11
- TS_PPS = 12

Example

```
# set
input:  0 M_SMAOUTPUT DISABLED
output: <OK>

# get
input:  0 M_SMAOUTPUT ?
output: 0 M_SMAOUTPUT DISABLED
```

M_SMASTATUS

code: 97

```
# get
<module-index> M_SMASTATUS ?
```

Description

For test modules with SMA connectors, this returns the status of the SMA input.

Actions

get

Parameters

1. `status`: *byte*, the status of the SMA input
   - `OK = 0`
   - `NO_VALID_SIGNAL = 1`

Example

```
# get
input:  0 M_SMASTATUS ?
output: 0 M_SMASTATUS OK
```

3.2.7 Impairment

M_EMULBYPASS

code: 454

```
# set
<module-index> M_EMULBYPASS <on_off>

# get
<module-index> M_EMULBYPASS ?
```

Description

Set emulator bypass mode. Emulator bypass mode will bypass the entire emulator for minimum latency.

Actions

set, get

Parameters

1. on_off: *byte*, the bypass mode of the network emulator.

- OFF = 0
- ON = 1

Example

```
# set
input:  0 M_EMULBYPASS OFF
output: <OK>

# get
input:  0 M_EMULBYPASS ?
output: 0 M_EMULBYPASS OFF
```

M_LATENCYMODE

code: 450

```
# set
<module-index> M_LATENCYMODE <mode>

# get
<module-index> M_LATENCYMODE ?
```

Description

Configures the latency mode for E100 Chimera module. In extended latency mode, the FPGA allows all latency parameters to be 10 times higher, at the cost of reduced latency precision.

Note: When change the latency mode, all latency configurations are reset on all ports in chimera module.

Actions

set, get

Parameters

1. mode: *byte*, specifying latency mode.

- NORMAL = 0
- EXTENDED = 1

Example

```
# set
input:  0 M_LATENCYMODE NORMAL
output: <OK>

# get
input:  0 M_LATENCYMODE ?
output: 0 M_LATENCYMODE NORMAL
```

The other module commands are the same as *Module Commands*.

3.2.8 Capability

M_CAPABILITIES

code: 89

```
# get
<module-index> M_CAPABILITIES ?
```

Description

Gets the module capabilities.

Actions

get

Parameters

1. `can_advanced_timing`: *integer*, is advanced timing functions supported?
   - `NO = 0`
   - `YES = 1`
2. `can_local_time_adjust`: *integer*, is local time adjustment supported?
   - `NO = 0`
   - `YES = 1`
3. `can_media_config`: *integer*, is module media configuration supported?
   - `NO = 0`
   - `YES = 1`
4. `require_multi_image`: *integer*, does this module switch images during runtime?
   - `NO = 0`
   - `YES = 1`
5. `is_chimera`: *integer*, is this a E100 Chimera module?
   - `NO = 0`
   - `YES = 1`
6. `max_clock_ppm`: *integer*, maximum supported absolute +clock ppm setting.
7. `can_tsn`: *integer*, does this module support Time Sensitive Networking (TSN) ?

- NO = 0
- YES = 1

8. can_ppm_sweep: *integer*, does this module support Local Clock Adjustment/Sweep (aka. PPM Sweep) ?
   - NO = 0
   - YES = 1
9. monitoring_bitmask: *integer*, extended module monitoring capabilities

Example

```
# get
input:  0 M_CAPABILITIES ?
output: 0 M_CAPABILITIES YES YES YES YES YES 400 NO YES 1
```

3.2.9 License

M_LICENSE_CWB_DETECTED

code: 402

```
# get
<module-index> M_LICENSE_CWB_DETECTED ?
```

Description

Returns if clock-windback is detected. If clock-windback has been detected the chassis is locked and no reservations of ports can be performed. To recover from clock-windback, set the system time correct (via the M4_SYSTEM_TIME command) and perform a license update (via the M_LICENSE_UPDATE command). Only applicable to L47 test module.

Actions

get

Parameters

1. `detected`: *byte*, whether clock-windback is detected
   - `NO = 0`
   - `YES = 1`

Example

```
# get
input:  0 M_LICENSE_CWB_DETECTED ?
output: 0 M_LICENSE_CWB_DETECTED NO
```

M_LICENSE_DEMO_INFO

`code: 400`

```
# get
<module-index> M_LICENSE_DEMO_INFO ?
```

Description

Returns info about the demo status of the module. Only applicable to L47 test module.

Actions

get

Parameters

1. **`demo`: *byte*, info of the demo status of the test module.**
   - `NON_DEMO = 0`
   - `DEMO = 1`
2. **`valid`: *byte*, info of the demo status of the test module.**
   - `INVALID = 0`
   - `VALID = 1`
3. **`permanent`: *byte*, info of the demo status of the test module.**
   - `NON_PERMANENT = 0`

- PERMANENT = 1

4. expiration: *long integer*, info of the demo status of the test module.

Example

```
# get
input:  0 M_LICENSE_DEMO_INFO ?
output: 0 M_LICENSE_DEMO_INFO NON_DEMO INVALID NON_PERMANENT␣
↪123456789123
```

M_LICENSE_LIST_BSON

code: 405

```
# get
<module-index> M_LICENSE_LIST_BSON ?
```

Description

Returns a list of locally stored licenses - formatted as a BSON document.

Actions

get

Parameters

1. bson: *hex list*, a list of locally stored licenses formatted as a BSON document.

Example

```
# get
input:  0 M_LICENSE_LIST_BSON ?
output: 0 M_LICENSE_LIST_BSON 0x57
```

M_LICENSE_MAINTENANCE_INFO

code: 401

```
# get
<module-index> M_LICENSE_MAINTENANCE_INFO ?
```

Description

Returns info about the maintenance license status for the module. Only applicable to L47 test module.

Actions

get

Parameters

1. `valid`: *byte*, the info about the maintenance license status for the test module
   - `INVALID = 0`
   - `VALID = 1`
2. `permanent`: *byte*, the info about the maintenance license status for the test module
   - `NON_PERMANENT = 0`
   - `PERMANENT = 1`
3. `expiration`: *long integer*, the info about the maintenance license status for the test module

Example

```
# get
input:  0 M_LICENSE_MAINTENANCE_INFO ?
output: 0 M_LICENSE_MAINTENANCE_INFO INVALID NON_PERMANENT 123456789123
```

M_LICENSE_ONLINE

code: 406

```
# set
<module-index> M_LICENSE_ONLINE <mode>

# get
<module-index> M_LICENSE_ONLINE ?
```

Description

Configures the chassis in online or offline mode. The online mode configuration defines two different license update procedures as described for the M_LICENSE_UPDATE command. In online mode the license update procedure requires access to the Internet. In offline mode the license update procedure can be performed without access to the Internet.

Actions

set, get

Parameters

1. mode: *byte*, the current online/offline mode of the L47 tester

- OFFLINE = 0
- ONLINE = 1

Example

```
# set
input:  0 M_LICENSE_ONLINE OFFLINE
output: <OK>

# get
input:  0 M_LICENSE_ONLINE ?
output: 0 M_LICENSE_ONLINE OFFLINE
```

M_LICENSE_UPDATE

code: 403

```
# set
<module-index> M_LICENSE_UPDATE
```

Description

This command instructs the chassis to update its local license information from FlexNet Operations. The chassis can be configured in on-line and off-line mode (by the M_LICENSE_ONLINE command). In on-line mode, the chassis sends a capability request to FlexNet Operations and receives a capability response. In offline mode a capability response (bin file) must be downloaded from FlexNet Operations and uploaded to the chassis. The capability response (bin file) is parsed and the license info is stored locally in trusted storage. A capability response (bin file) has a lifetime of one day (24 hours). The result of the license update operation can be retrieved by M_LICENSE_UPDATE_STATUS.

Actions

set

Example

```
# set
input:  0 M_LICENSE_UPDATE
output: <OK>
```

M_LICENSE_UPDATE_STATUS

code: 404

```
# get
<module-index> M_LICENSE_UPDATE_STATUS ?
```

Description

Returns the status of the latest license update operations.

Actions

get

Parameters

1. `update_state`: *byte*, the status of the latest license update operation
   - `NONE = 0`
   - `UPDATING = 1`
   - `UPDATE_SUCCESS = 2`
   - `UPDATE_FAIL = 3`
2. `last_update`: *long integer*, the status of the latest license update operation
3. `last_success`: *long integer*, the status of the latest license update operation
4. `last_fail`: *long integer*, the status of the latest license update operation
5. `info`: *string*, the status of the latest license update operation

Example

```
# get
input:  0 M_LICENSE_UPDATE_STATUS ?
output: 0 M_LICENSE_UPDATE_STATUS NONE 0 0 0 0 "this is the info"
```

3.2.10 Misc

M_FPGAREIMAGE

`code:  91`

```
# set
<module-index> M_FPGAREIMAGE
```

Description

Reload FPGA image.

Actions

set

Parameters

1. key_code: *integer*, must be 42.

Example

```
# set
input:  0 M_FPGAREIMAGE
output: <OK>
```

M_MULTIUSER

code: 92

```
# set
<module-index> M_MULTIUSER <on_off>

# get
<module-index> M_MULTIUSER ?
```

Description

Enable or disable multiple sessions to control the same module.

Actions

set, get

Parameters

1. `on_off`: *byte*, Enable or disable multiple sessions to control the same module
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0 M_MULTIUSER OFF
output: <OK>

# get
input:  0 M_MULTIUSER ?
output: 0 M_MULTIUSER OFF
```

M_UPGRADE

`code:  81`

```
# set
<module-index> M_UPGRADE <magic> <image_name>
```

Description

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.

Actions

set

Parameters

1. `magic`: *integer*, -1480937026 (0x58454E41)
2. `image_name`: *string*, the fully qualified name of a file previously uploaded to the chassis

Example

```
# set
input:  0 M_UPGRADE -1480937026 "image.name"
output: <OK>
```

M_UPGRADEPAR

code: 87

```
# set
<module-index> M_UPGRADEPAR <magic> <image_name>
```

Description

Parallel module upgrade.

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.

Actions

set

Parameters

1. `magic`: *integer*, -1480937026 (0x58454E41)
2. `image_name`: *string*, the fully qualified name of a file previously uploaded to the chassis

Example

```
# set
input:  0 M_UPGRADEPAR -1480937026 "image.name"
output: <OK>
```

M_UPGRADEPROGRESS

code: 82

```
# get
<module-index> M_UPGRADEPROGRESS ?
```

Description

Provides a value indicating the current stage of an ongoing 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.

Actions

get

Parameters

1. progress: *integer*,

Example

```
# get
input:  0 M_UPGRADEPROGRESS ?
output: 0 M_UPGRADEPROGRESS 1
```

3.2.11 Full Config

M_CONFIG

code: 74

```
# get
0 M_CONFIG ?
```

Description

Return all setable module values

Actions

get

Parameters

Example

```
# get
input:  0 M_CONFIG ?
```

3.3 Port

Port commands that deal with basic information about, and configuration of test ports. The port command names all have the form P_<xxx> and require a module index id and a port index id. In general, port commands cannot be changed while traffic is on. Additionally, every stream must be disabled before changing parameters that affect the bandwidth of the port.

3.3.1 General

Identification

P_COMMENT

code: 112

```
# set
<module-index>/<port-index> P_COMMENT <comment>

# get
<module-index>/<port-index> P_COMMENT ?
```

Description

The description of a port.

Actions

set, get

Parameters

1. `comment`: *string*, the description of the port

Example

```
# set
input:  0/1 P_COMMENT "This is a comment"
output: <OK>

# get
input:  0/1 P_COMMENT ?
output: 0/1 P_COMMENT "This is a comment"
```

P_INTERFACE

`code:  107`

```
# get
<module-index>/<port-index> P_INTERFACE ?
```

Description

Obtains the name of the physical interface type of a port.

Actions

get

Parameters

1. `interface`: *string*, the name of the physical interface type of a port.

Example

```
# get
input:  0/1 P_INTERFACE ?
output: 0/1 P_INTERFACE "SFP-E 10/100/1000M [Triple] [Auto]"
```

Control

P_FLASH

`code:  123`

```
# set
<module-index>/<port-index> P_FLASH <on_off>

# get
<module-index>/<port-index> P_FLASH ?
```

Description

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.

Actions

set, get

Parameters

1. on_off: *byte*, the status of the LED flashing status of the port.

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P_FLASH OFF
output: <OK>

# get
input:  0/1 P_FLASH ?
output: 0/1 P_FLASH OFF
```

P_RESET

code: 104

```
# set
<module-index>/<port-index> P_RESET
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 P_RESET
output: <OK>
```

Status

P_RESERVATION

code: 102

```
# set
<module-index>/<port-index> P_RESERVATION <operation>

# get
<module-index>/<port-index> P_RESERVATION ?
```

Description

You set this command 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.

Actions

set, get

Parameters

1. `operation`: *byte*, (when set) the reservation of the test port, i.e., reserve, release, or relinquish.

- `RELEASE = 0`
- `RESERVE = 1`
- `RELINQUISH = 2`

2. `operation`: *byte*, (when get) containing the operation to perform. The reservation parameters are asymmetric with respect to set/get. When set, it contains the operation to perform. When get, it contains the status.

- `RELEASED = 0`
- `RESERVED_BY_YOU = 1`
- `RESERVED_BY_OTHER = 2`

Example

```
# set
input:  0/1 P_RESERVATION RELEASE
output: <OK>

# get
input:  0/1 P_RESERVATION ?
output: 0/1 P_RESERVATION RESERVED_BY_YOU
```

P_RESERVEDBY

code: 103

```
# get
<module-index>/<port-index> P_RESERVEDBY ?
```

Description

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`.

Actions

get

Parameters

1. `username`: *string*, the username of the user who has the port reserved.

Example

```
# get
input:  0/1 P_RESERVEDBY ?
output: 0/1 P_RESERVEDBY 'peter-parker'
```

P_STATUS

code: 303

```
# get
<module-index>/<port-index> P_STATUS ?
```

Description

Get the received signal level for optical ports.

Actions

get

Parameters

1. `optical_power`: *integer list*, the received signal level for optical ports, in nanowatts, -1 when not available

Example

```
# get
input:  0/1 P_STATUS ?
output: 0/1 P_STATUS 0 1
```

Capability

P_CAPABILITIES

code: 106

```
# get
<module-index>/<port-index> P_CAPABILITIES ?
```

Description

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

Actions

get

Parameters

1. `max_speed`: *integer*, maximum wire speed in Mbps, for fastest transceiver and mode. This is L1 Mbps, related to calculating `PS_RATEL2BPS` and `PS_RATEPPS`.
2. `max_speed_reduction`: *integer*, maximum ppm value of speed reduction for `P_SPEEDREDUCTION`.
3. `min_interframe_gap`: *integer*, minimum bytes between frames for `P_INTERFRAMEGAP`.
4. `max_interframe_gap`: *integer*, maximum explicit bytes between frames for `P_INTERFRAMEGAP`.
5. `max_preamble`: *integer*, maximum preamble bytes included in frame. Ethernet preamble byte count, include in the IFG byte count.
6. `max_streams_per_port`: *integer*, maximum streams per port. Max length of `PS_INDICES: stream_indices`, and `PS_CREATE` max index value.
7. `max_percent`: *integer*, maximum input rate in percent. Maximum incoming traffic rate in percent.
8. `max_pps`: *integer*, maximum input rate in pps. Maximum incoming traffic rate in pps.
9. `max_mbps`: *integer*, maximum input rate in Mbps. Maximum incoming traffic rate in Mbps.
10. `max_seed`: *integer*, maximum random seed for `P_RANDOMSEED: seed`.
11. `max_tx_packet_limit`: *integer*, maximum stop-after-n-packet limitation for `P_TXPACKETLIMIT: packet_count_limit`.
12. `max_burst_size`: *integer*, maximum packets per burst for `PS_BURST: size`.
13. `min_packet_length`: *integer*, minimum bytes in total packet for `PS_PACKETLENGTH`.
14. `max_packet_length`: *integer*, maximum bytes in total packet for `PS_PACKETLENGTH`.
15. `max_header_length`: *integer*, maximum bytes in auto-generated packet header for `P_MAXHEADERLENGTH`.
16. `max_protocol_segments`: *integer*, maximum number of protocol segments. For `PS_HEADERPROTOCOL: segments` and it should be the same as `C_CAPABILITIES: max_protocol_count`.

17. `max_pattern_length`: *integer*, maximum bytes in payload pattern. For `PS_PAYLOAD: hex_data` max length.
18. `max_modifiers`: *integer*, maximum 16-bit modifiers per stream. For `PS_MODIFIERCOUNT: modifier_count`.
19. `max_modifier_bytes`: *integer*, maximum bytes in a modified field. Fixed to 2 bytes.
20. `max_repeat`: *integer*, maximum repeats for a modifier. For `PS_MODIFIER: repetition`.
21. `max_tpid`: *integer*, maximum TPLD ID value. For `PS_TPLDID: test_payload_identifier`.
22. `max_manual_packets`: *integer*, maximum manual packets. Not in use.
23. `max_match_terms`: *integer*, max filter match terms per port. Max length of `PM_INDICES: match_term_xindices`, and `PM_CREATE` max index value.
24. `max_length_terms`: *integer*, max filter length terms per port. Max length of `PL_INDICES: length_term_xindices`, and `PL_CREATE` max index value.
25. `max_ors`: *integer*, max or-terms per filter.
26. `max_nots`: *integer*, max or-terms with nots per filter.
27. `max_filters`: *integer*, max filters per port. Max length of `PF_INDICES: filter_xindices`, and `PF_CREATE` max index value.
28. `max_captured_packets`: *integer*, max captured packets at one time. `PC_STATS: packets` max value
29. `max_tpld_stats`: *integer*, max number of different TPLDs for RX statistics. Max TPLD index a RX port can handle.
30. `max_histogram`: *integer*, max number of sampled histograms. Max length of `PD_INDICES: histogram_indices`, and `PD_CREATE` max index value.
31. `max_32bit_modifiers`: *integer*, max 32-bit modifiers per stream. Max value of `PS_MODIFIEREXTCOUNT: ext_modifier_count`.
32. `can_set_autoneg`: *integer*, whether supports auto negotiation. For `P_AUTONEGSELECTION`, and it tells whether the port is a RJ45 port or not.
33. `can_tcp_checksum`: *integer*, whether supports TCP with valid checksum. `PS_HEADERPROTOCOL: segments = TCPCHECK` supported or not.
34. `can_udp_checksum`: *integer*, whether supports UDP with valid checksum. `PS_HEADERPROTOCOL: segments = UDPCHECK` supported or not.
35. `can_eee`: *integer*, whether supports Energy Efficient Ethernet. Corresponds to whether the following commands are supported: `P_LPSUPPORT`, `P_LPRXPOWER`, `P_LPSNRMARGIN`, `P_LPPARTNERAUTONEG`, `P_LPSTATUS`, `P_LPTXMODE`, and `P_LPENABLE`.
36. `can_hw_reg_access`: *integer*, whether supports hardware register access. Corresponds to whether `PX_RW` is supported.

37. `can_tcvr_mii_reg_access`: *integer*, whether supports transceiver MII access. Corresponds to whether `PX_MII` is supported.
38. `can_adv_phy_man`: *integer*, whether supports advanced PHY management. Corresponds to whether the following commands are supported: `PP_PHYTXEQ`, `PP_PHYRXEQ`, `PP_PHYRETUNE`, `PP_PHYAUTOTUNE`, and `PP_PHYSIGNALSTATUS`.
39. `can_micro_tpld`: *integer*, whether supports micro TPLD. Whether `P_TPLDMODE: mode = MICRO` is supported.
40. `can_mdi_mdix`: *integer*, whether supports MDI/MDIX. Whether `P_MDIXMODE` is supported.
41. `can_payload_mode`: *integer*, whether supports payload mode. Whether `P_PAYLOADMODE` is supported.
42. `can_custom_data_fields`: *integer*, whether supports custom data fields. Whether `P_PAYLOADMODE: mode = CDF` is supported.
43. `can_ext_payload`: *integer*, whether supports extended payload. Whether `P_PAYLOADMODE: mode = EXTPL` is supported.
44. `can_dyn_traffic_change`: *integer*, hether supports dynamic traffic change. Whether `P_DYNAMIC: on_off = ON` is supported.
45. `can_sync_traffic_start`: *integer*, whether supports synchronized traffic start. Whether `C_TRAFFICSYNC` and `C_TRAFFIC` are supported or not.
46. `can_pfc`: *integer*, whether supports Priority Flow Control. Whether `P_PFCENABLE` is supported.
47. `can_pcs_pma_config`: *integer*, whether this port can provide PCS/PMA configuration and status. Corresponds to whether the following commands are supported: `PP_TXLANEINJECT`, `PP_TXERRORRATE`, `PP_TXLANECONFIG`, `PP_PRBSCONFIG`, and `PP_PHYSETTINGS`.
48. `can_fec`: *integer*, for `PP_FECMODE`. This value is a bit map: [0] = RS FEC KR, [1] = RS FEC KP, [2] = FC FEC, [3] = RS FEC Int [31] = Mandatory. Position [0] and [1] are mutually exclusive. If [31] is set, the port does not support `OFF`. If [0] is set, the port supports `ON`, and supposedly `RS_FEC` and `RS_FEC_KR`. If [1] is set, the port supports `ON`, and supposedly `RS_FEC` and `RS_FEC_KP`. If [2] is set, the port supports `FC_FEC`. If [3] is set, the port supports `RS_FEC_INT`
49. `can_fec_stats`: *integer*, can this port provide advanced FEC stats of type x? [0] = symbol error distribution. For `PP_RXTOTALSTATS`.
50. `can_tx_eq`: *integer*, whether supports TX EQ settings `PP_PHYTXEQ`.
51. `can_rx_retune`: *integer*, whether supports RX retuning `PP_PHYRETUNE`.
52. `prbs_types_supported`: *integer*, type of PRBS supported [0] = lane-based, [1] = PHY-based, [2-31] = reserved. For `PP_PRBSTYPE: prbs_inserted_type`.

See also:

Detailed explanation in *Explanation of prbs_types_supported*

53. `prbs_inversions_supported`: *integer*, [0] = lane-based supports inv, [1] = PHY-based supports inv, [2-31] = reserved. For `PP_PRBSTYPE: invert`.
54. `prbs_polys_supported`: *integer list*, bit map for each PRBS type. For `PP_PRBSTYPE: polynomial`.

See also:

Detailed explanation in *Explanation of prbs_polys_supported*

55. `serdes_count`: *integer*, number of physical serdes on line-side.
56. `lane_count`: *integer*, number of lanes (virtual).
57. `tx_eq_tap_count`: *integer*, number of TXEQ taps.
58. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
59. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
60. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
61. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
62. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
63. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
64. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
65. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
66. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
67. `tx_eq_tap_max_val`: *integer*, max value of individual TXEQ tap.
68. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
69. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
70. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
71. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
72. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
73. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
74. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
75. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
76. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.
77. `tx_eq_tap_min_val`: *integer*, min value of individual TXEQ tap.

Note: This table shows the meaning of #58 to #77 based on the module type and modulation.

#	Z800 Freya (112G serdes)	Z400 Thor (56G serdes, PAM4)	Z400 Thor (56G serdes, NRZ)	Z100 Loki (28G serdes, NRZ)
58	max value of pre	max value of pre	max value of pre	max value of pre
59	max value of main	max value of main	max value of main	max value of main
60	max value of post	max value of post	max value of post	max value of post
61	max value of pre2	max value of pre2	max value of pre2	0
62	max value of pre3	max value of post2	max value of post2	0
63	0	max value of post3	max value of post3	0
64	0	0	0	0
65	0	0	0	0
66	0	0	0	0
67	0	0	0	0
68	min value of pre	min value of pre	min value of pre	min value of pre
69	min value of main	min value of main	min value of main	min value of main
70	min value of post	min value of post	min value of post	min value of post
71	min value of pre2	min value of pre2	min value of pre2	0
72	min value of pre3	min value of post2	min value of post2	0
73	0	min value of post3	min value of post3	0
74	0	0	0	0
75	0	0	0	0
76	0	0	0	0
77	0	0	0	0

78. `max_fec_correctable_symbol_count`: *integer*, max number of symbols correctable by the current FEC.

79. `max_xmit_one_packet_length`: *integer*, maximum size (in bytes) of packets, which can be sent using `P_XMITONE` (replay/streaming interface). For `P_XMITONE: hex_data` length.

80. `tx_runt_packet_min_length`: *integer*, minimum TX packet size supported by runt block. Zero = not supported. For `P_TXRUNTLENGTH: runt_length`.

81. `rx_runt_packet_min_length`: *integer*, minimum RX packet size supported by runt block. Zero = not supported. For `P_RXRUNTLENGTH: runt_length`.

82. `can_manipulate_preamble`: *integer*, whether this port can manipulate the preamble. Whether `P_TXPREAMBLE_REMOVE` and `P_RXPREAMBLE_INSERT` are supported.

83. `can_set_link_train`: *integer*, whether this port can set link training `PP_LINKTRAIN`.

84. `can_link_flap`: *integer*, whether this port supports link flap. Whether `PP_LINKFLAP_ENABLE` and `PP_LINKFLAP_PARAMS` are supported.

85. `can_auto_neg_base_r`: *integer*, whether the port currently can perform BASE-R autonegotiation (as opposed to RJ45 BASE-T). Whether `PP_AUTONEG` is supported.

86. `can_pma_error_pulse`: *integer*, whether this port supports *PMA pulse error injection*.

Whether PP_PMAERRPUL_ENABLE and PP_PMAERRPUL_PARAMS are supported.

87. is_chimera: *integer*, whether this is a E100 Chimera port.
88. has_p2p_loop_partner: *integer*, whether this port currently has a port-to-port loop partner. Whether P_LOOPBACK: mode = PORT2PORT is supported.
89. p2p_loop_partner: *integer*, The port-to-port loop partner for the port. -1 means N/A. If this is -1, P_LOOPBACK: mode cannot be PORT2PORT.
90. traffic_engine: *integer*, the enabled traffic engine on port. 1 = TGA, 2 = uTGA (micro TGA).
91. reconc_sublayer: *integer*, the Reconciliation Sublayer support
    - NO_SUPPORT = 0
    - FAULT_SIGNALING = 1, which means P_FAULTSTATUS and P_FAULTSIGNALLING are supported by the port.
92. max_match_term_pos: *integer*, max match term position in bytes
93. stream_misc: *integer*, Bit pattern, what streams on this port can do. [0]: Whether the port supports streams with DEC8/INC16/DEC16 payload. [1]: Whether the port supports INCPLDFROM0 stream option (refer to the PS_OPTIONS command).
94. rxeq_cap_ctle_low_min: *integer*, min value of CTLE LOW.
95. rxeq_cap_ctle_high_min: *integer*, min value of CTLE HIGH.
96. rxeq_cap_agc_min: *integer*, min value of Automatic Gain Control.
97. rxeq_cap_oc_min: *integer*, min value of Offset Cancellation.
98. rxeq_cap_cdr_min: *integer*, min value of CDR, always 0.
99. rxeq_cap_ffe_pre1_min: *integer*, min value of FFE Pre 1.
100. rxeq_cap_ffe_pre2_min: *integer*, min value of FFE Pre 2.
101. rxeq_cap_ffe_pre3_min: *integer*, min value of FFE Pre 3.
102. rxeq_cap_ffe_pre4_min: *integer*, min value of FFE Pre 4.
103. rxeq_cap_ffe_pre5_min: *integer*, min value of FFE Pre 5.
104. rxeq_cap_ffe_pre6_min: *integer*, min value of FFE Pre 6.
105. rxeq_cap_ffe_pre7_min: *integer*, min value of FFE Pre 7.
106. rxeq_cap_ffe_pre8_min: *integer*, min value of FFE Pre 8.
107. rxeq_cap_dfe_min: *integer*, min value of DFE, always 0.
108. rxeq_cap_ffe_post1_min: *integer*, min value of FFE Post 1.
109. rxeq_cap_ffe_post2_min: *integer*, min value of FFE Post 2.
110. rxeq_cap_ffe_post3_min: *integer*, min value of FFE Post 3.
111. rxeq_cap_ffe_post4_min: *integer*, min value of FFE Post 4.

112. `rxeq_cap_ffe_post5_min`: *integer*, min value of FFE Post 5.
113. `rxeq_cap_ffe_post6_min`: *integer*, min value of FFE Post 6.
114. `rxeq_cap_ffe_post7_min`: *integer*, min value of FFE Post 7.
115. `rxeq_cap_ffe_post8_min`: *integer*, min value of FFE Post 8.
116. `rxeq_cap_ffe_post9_min`: *integer*, min value of FFE Post 9.
117. `rxeq_cap_ffe_post10_min`: *integer*, min value of FFE Post 10.
118. `rxeq_cap_ffe_post11_min`: *integer*, min value of FFE Post 11.
119. `rxeq_cap_ffe_post12_min`: *integer*, min value of FFE Post 12.
120. `rxeq_cap_ffe_post13_min`: *integer*, min value of FFE Post 13.
121. `rxeq_cap_ffe_post14_min`: *integer*, min value of FFE Post 14.
122. `rxeq_cap_ffe_post15_min`: *integer*, min value of FFE Post 15.
123. `rxeq_cap_ffe_post16_min`: *integer*, min value of FFE Post 16.
124. `rxeq_cap_ffe_post17_min`: *integer*, min value of FFE Post 17.
125. `rxeq_cap_ffe_post18_min`: *integer*, min value of FFE Post 18.
126. `rxeq_cap_ffe_post19_min`: *integer*, min value of FFE Post 19.
127. `rxeq_cap_ffe_post20_min`: *integer*, min value of FFE Post 20.
128. `rxeq_cap_ffe_post21_min`: *integer*, min value of FFE Post 21.
129. `rxeq_cap_ffe_post22_min`: *integer*, min value of FFE Post 22.
130. `rxeq_cap_ffe_post23_min`: *integer*, min value of FFE Post 23.
131. `reserved`: *integer*, reserved.
132. `reserved`: *integer*, reserved.
133. `reserved`: *integer*, reserved.
134. `reserved`: *integer*, reserved.
135. `reserved`: *integer*, reserved.
136. `reserved`: *integer*, reserved.
137. `reserved`: *integer*, reserved.
138. `reserved`: *integer*, reserved.
139. `reserved`: *integer*, reserved.
140. `reserved`: *integer*, reserved.
141. `rxeq_cap_ctle_low_max`: *integer*, max value of CTLE LOW.
142. `rxeq_cap_ctle_high_max`: *integer*, max value of CTLE HIGH.
143. `rxeq_cap_agc_max`: *integer*, max value of Automatic Gain Control.

144. `rxeq_cap_oc_max`: *integer*, max value of Offset Cancellation.
145. `rxeq_cap_cdr_max`: *integer*, max value of CDR, always 0.
146. `rxeq_cap_ffe_pre1_max`: *integer*, max value of FFE Pre 1.
147. `rxeq_cap_ffe_pre2_max`: *integer*, max value of FFE Pre 2.
148. `rxeq_cap_ffe_pre3_max`: *integer*, max value of FFE Pre 3.
149. `rxeq_cap_ffe_pre4_max`: *integer*, max value of FFE Pre 4.
150. `rxeq_cap_ffe_pre5_max`: *integer*, max value of FFE Pre 5.
151. `rxeq_cap_ffe_pre6_max`: *integer*, max value of FFE Pre 6.
152. `rxeq_cap_ffe_pre7_max`: *integer*, max value of FFE Pre 7.
153. `rxeq_cap_ffe_pre8_max`: *integer*, max value of FFE Pre 8.
154. `rxeq_cap_dfe_max`: *integer*, max value of DFE, always 0.
155. `rxeq_cap_ffe_post1_max`: *integer*, max value of FFE Post 1.
156. `rxeq_cap_ffe_post2_max`: *integer*, max value of FFE Post 2.
157. `rxeq_cap_ffe_post3_max`: *integer*, max value of FFE Post 3.
158. `rxeq_cap_ffe_post4_max`: *integer*, max value of FFE Post 4.
159. `rxeq_cap_ffe_post5_max`: *integer*, max value of FFE Post 5.
160. `rxeq_cap_ffe_post6_max`: *integer*, max value of FFE Post 6.
161. `rxeq_cap_ffe_post7_max`: *integer*, max value of FFE Post 7.
162. `rxeq_cap_ffe_post8_max`: *integer*, max value of FFE Post 8.
163. `rxeq_cap_ffe_post9_max`: *integer*, max value of FFE Post 9.
164. `rxeq_cap_ffe_post10_max`: *integer*, max value of FFE Post 10.
165. `rxeq_cap_ffe_post11_max`: *integer*, max value of FFE Post 11.
166. `rxeq_cap_ffe_post12_max`: *integer*, max value of FFE Post 12.
167. `rxeq_cap_ffe_post13_max`: *integer*, max value of FFE Post 13.
168. `rxeq_cap_ffe_post14_max`: *integer*, max value of FFE Post 14.
169. `rxeq_cap_ffe_post15_max`: *integer*, max value of FFE Post 15.
170. `rxeq_cap_ffe_post16_max`: *integer*, max value of FFE Post 16.
171. `rxeq_cap_ffe_post17_max`: *integer*, max value of FFE Post 17.
172. `rxeq_cap_ffe_post18_max`: *integer*, max value of FFE Post 18.
173. `rxeq_cap_ffe_post19_max`: *integer*, max value of FFE Post 19.
174. `rxeq_cap_ffe_post20_max`: *integer*, max value of FFE Post 20.
175. `rxeq_cap_ffe_post21_max`: *integer*, max value of FFE Post 21.

176. `rxeq_cap_ffe_post22_max`: *integer*, max value of FFE Post 22.
177. `rxeq_cap_ffe_post23_max`: *integer*, max value of FFE Post 23.
178. `reserved`: *integer*, reserved.
179. `reserved`: *integer*, reserved.
180. `reserved`: *integer*, reserved.
181. `reserved`: *integer*, reserved.
182. `reserved`: *integer*, reserved.
183. `reserved`: *integer*, reserved.
184. `reserved`: *integer*, reserved.
185. `reserved`: *integer*, reserved.
186. `reserved`: *integer*, reserved.
187. `reserved`: *integer*, reserved.
188. `length_histogram_step_min`: *integer*, minimum step size for length histograms.
189. `length_histogram_step_max`: *integer*, maximum step size for length histograms.
190. `latency_histogram_step_min`: *integer*, minimum step size for latency histograms.
191. `latency_histogram_step_max`: *integer*, maximum step size for latency histograms.
192. `tcvr_i2c_min_freq_khz`: *integer*, Minimum configurable transceiver I2C frequency in kHz. Default is 100 kHz
193. `tcvr_i2c_max_freq_khz`: *integer*, Maximum configurable transceiver I2C frequency in kHz. Default is 100 kHz; value is either based on module HW limits or transceiver capability limits
194. `can_eyescan`: *integer*, bitmask, Bit 0: Sampled Eyescan supported. Reserved, but not implemented yet: Bit 1: Statistical Eye Scan supported (the Z100 Loki eye-scan).

Example

```
# get
input:  0/1 P_CAPABILITIES ?
```

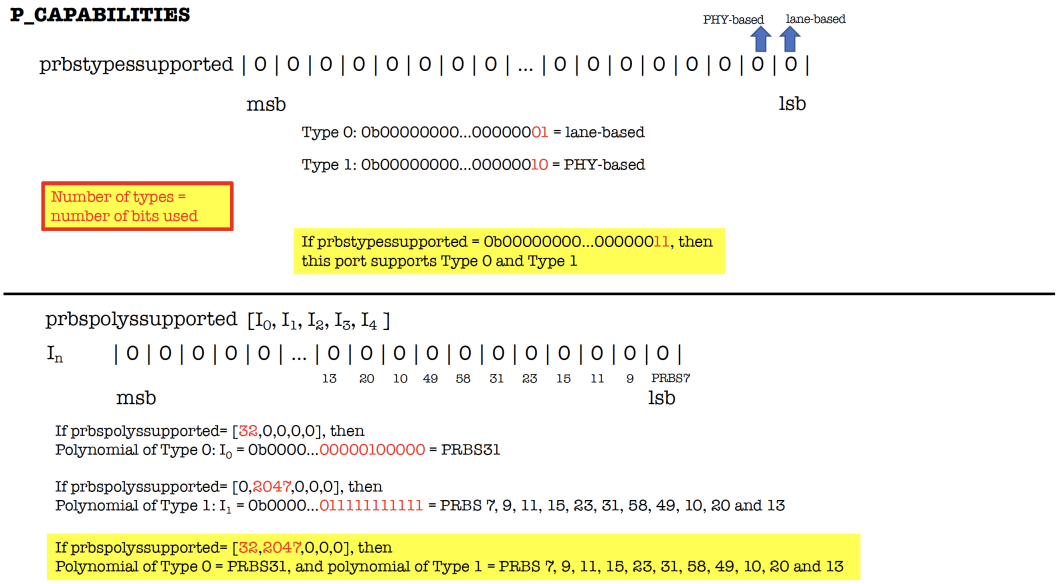


Fig. 3.1: Explanation of `prbs_types_supported` and `prbs_polys_supported`

P_CAPABILITIES_EXT

`code: 423`

```
# get
<module-index>/<port-index> P_CAPABILITIES_EXT ?
```

Description

Returns a JSON string specifying various internal limits of a port.

Actions

get

Parameters

`json`: *string*, JSON string specifying various internal limits of a port.

The JSON schema:

```
{
    "properties": {
        "max_speed_mbps": 100000,
        "max_speed_reduction_ppm": 1000,
        "interframe_gap_bytes": {
```

(continues on next page)

(continued from previous page)


```
            "min": 16,
            "max": 63
        },
        "max_preamble_bytes": 12,
        "max_streams": 256,
        "max_percent": 200,
        "max_pps": 390625000,
        "max_mbps": 200000,
        "max_seed": 1000000000,
        "max_tx_packet_limit": 2147483647,
        "max_burst_size_bytes": 2000000000,
        "packet_length_bytes": {
            "min": 48,
            "max": 12288
        },
        "max_header_length_bytes": 2048,
        "max_protocol_segments": 30,
        "max_pattern_length_bytes": 18,
        "modifier": {
            "max_count": 8,
            "max_bytes": 2,
            "max_repeat": 4096,
            "max_32bit_count": 4
        },
        "max_tpid": 2015,
        "max_manual_packets": 10,
        "filter": {
            "max_match_terms": 6,
            "max_length_terms": 6,
            "max_match_term_pos": 2032,
            "max_ors": 4,
            "max_nots": 2,
            "max_filters": 6
        },
        "max_captured_packets": 4096,
        "max_tpld_stats": 2015,
        "histogram": {
            "max_counts": 2,
            "length_step_min": 1,
            "length_step_max": 2048,
            "latency_step_min": 16,
            "latency_step_max": 2097152
        },
        "can_autoneg": false,
        "can_tcp_checksum": true,
        "can_udp_checksum": true,
```


(continues on next page)

(continued from previous page)


```
        "can_eee": false,
        "can_hw_reg_access": true,
        "can_tcvr_mii_reg_access": false,
        "can_adv_phy_man": true,
        "can_micro_tpld": false,
        "can_mdi_mdix": false,
        "payload": {
            "can_payload_mode": true,
            "can_custom_data_fields": true,
            "can_ext_payload": true
        },
        "can_dynamic_traffic": true,
        "can_sync_traffic_start": true,
        "can_pfc": true,
        "can_pcs_pma": true,
        "fec": {
            "schemes_supported_bitmask": [],
            "stats_supported_bitmask": [],
            "max_correctable_symbols": 0
        },
        "can_rx_retune": true,
        "prbs": {
            "lane_based": {
                "is_supported": true,
                "inversion_supported": false,
                "polys": [
                    {
                        "name": "PRBS31",
                        "bxmp_enum": 32
                    }
                ]
            },
            "phy_based": {
                "is_supported": false,
                "inversion_supported": false,
                "polys": []
            }
        },
        "serdes_count": 1,
        "lane_count": 1,
        "tx_eq": {
            "is_supported": true,
            "tap_count": 3,
            "taps": [
                {
                    "name": "pre1",
```


(continues on next page)

(continued from previous page)


```
                    "min": -31,
                    "max": 31
                },
                {
                    "name": "main",
                    "min": 0,
                    "max": 31
                },
                {
                    "name": "post",
                    "min": -31,
                    "max": 31
                }
            ]
        },
        "max_xmit_one_packet_length_bytes": 2048,
        "runt": {
            "tx_packet_min_length_bytes": 0,
            "rx_packet_min_length_bytes": 0
        },
        "can_manipulate_preamble": false,
        "can_link_train": false,
        "can_link_flap": false,
        "can_auto_neg_base_r": false,
        "pma_error_type_bitmask": [],
        "is_chimera": false,
        "port2port_loop": {
            "has_partner": true,
            "partner_idx": 1
        },
        "traffic_engine": 1,
        "reconc_sublayer": {
            "supported_bitmask": []
        },
        "stream_option_supported_bitmask": [],
        "rxeq_ext": {
            "capabilities": []
        },
        "tcvr_i2c_speed": {
            "min_speed_khz": 100,
            "max_speed_khz": 100
        },
        "eyescan": {
            "can_sampled_eyescan": true
        },
    }
```


(continues on next page)

(continued from previous page)

```
}
```

Example

```
# get
input:  0/1 P_CAPABILITIES_EXT ?
```

3.3.2 Traffic Generation

Speed

P_SPEED

code: 110

```
# get
<module-index>/<port-index> P_SPEED ?
```

Description

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

Actions

get

Parameters

1. `port_speed`: *integer*, the current physical speed of the port's interface.

Example

```
# get
input:  0/1 P_SPEED ?
output: 0/1 P_SPEED 1
```

P_SPEEDREDUCTION

code: 113

```
# set
<module-index>/<port-index> P_SPEEDREDUCTION <ppm>

# get
<module-index>/<port-index> P_SPEEDREDUCTION ?
```

Description

A speed reduction applied to the transmitting 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.

Actions

set, get

Parameters

1. ppm: *integer*, the speed reduction ppm value of the test port

Example

```
# set
input:  0/1 P_SPEEDREDUCTION 1
output: <OK>

# get
input:  0/1 P_SPEEDREDUCTION ?
output: 0/1 P_SPEEDREDUCTION 1
```

P_SPEEDS_SUPPORTED

code: 396

```
# get
<module-index>/<port-index> P_SPEEDS_SUPPORTED ?
```

Description

Read the speeds supported by the port. The speeds supported by a port depends on the transceiver inserted into the port. A series of 0/1 values, identifying which speeds are supported by the port.

Note: Ports can support zero (in case of e.g. empty cage), one, or multiple speeds.

Actions

get

Parameters

1. `auto`: *short integer*, the speeds supported by the port
2. `f10M`: *short integer*, the speeds supported by the port
3. `f100M`: *short integer*, the speeds supported by the port
4. `f1G`: *short integer*, the speeds supported by the port
5. `f10G`: *short integer*, the speeds supported by the port
6. `f40G`: *short integer*, the speeds supported by the port
7. `f100G`: *short integer*, the speeds supported by the port
8. `f10MHDX`: *short integer*, the speeds supported by the port
9. `f100MHDX`: *short integer*, the speeds supported by the port
10. `f10M100M`: *short integer*, the speeds supported by the port
11. `f100M1G`: *short integer*, the speeds supported by the port
12. `f100M1G10G`: *short integer*, the speeds supported by the port
13. `f2500M`: *short integer*, the speeds supported by the port
14. `f5G`: *short integer*, the speeds supported by the port
15. `f100M1G2500M`: *short integer*, the speeds supported by the port

16. `f25G`: *short integer*, the speeds supported by the port
17. `f50G`: *short integer*, the speeds supported by the port
18. `f200G`: *short integer*, the speeds supported by the port
19. `f400G`: *short integer*, the speeds supported by the port
20. `f800G`: *short integer*, the speeds supported by the port
21. `f1600G`: *short integer*, the speeds supported by the port

Example

```
# get
input:  0/1 P_SPEEDS_SUPPORTED ?
output: 0/1 P_SPEEDS_SUPPORTED 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0␣
↪0
```

P_SPEEDSELECTION

code: 109

```
# set
<module-index>/<port-index> P_SPEEDSELECTION <mode>

# get
<module-index>/<port-index> P_SPEEDSELECTION ?
```

Description

The speed mode of an autoneg port with an interface type supporting multiple speeds.

Note: This is only a settable command when speed is selected at the port level. Use the *M_CFPCONFIG* command when speed is selected at the module level.

Actions

set, get

Parameters

1. `mode`: *byte*, the speed mode of the port with an interface type supporting multiple speeds

- `AUTO = 0`
- `F10M = 1`
- `F100M = 2`
- `F1G = 3`
- `F10G = 4`
- `F40G = 5`
- `F100G = 6`
- `F10MHDX = 7`
- `F100MHDX = 8`
- `F10M100M = 9`
- `F100M1G = 10`
- `F100M1G10G = 11`
- `F2500M = 12`
- `F5G = 13`
- `F100M1G2500M = 14`
- `F25G = 15`
- `F50G = 16`
- `F200G = 17`
- `F400G = 18`
- `F800G = 19`
- `F1600G = 20`

Example

```
# set
input:  0/1 P_SPEEDSELECTION AUTO
output: <OK>

# get
input:  0/1 P_SPEEDSELECTION ?
output: 0/1 P_SPEEDSELECTION AUTO
```

TX Control

P_DYNAMIC

code: 368

```
# set
<module-index>/<port-index> P_DYNAMIC <on_off>

# get
<module-index>/<port-index> P_DYNAMIC ?
```

Description

Controls if a >10G port supports dynamic changes when the traffic is
running. This command is only supported by ports >10G.

Actions

set, get

Parameters

1. on_off: *byte*, whether the port should support dynamic changes when the traffic is running

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P_DYNAMIC OFF
output: <OK>

# get
input:  0/1 P_DYNAMIC ?
output: 0/1 P_DYNAMIC OFF
```

P_RECEIVESYNC

code: 111

```
# get
<module-index>/<port-index> P_RECEIVESYNC ?
```

Description

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

Actions

get

Parameters

1. sync_status: *byte*, the current in-sync status for a port's receive interface.
   - NO_SYNC = 0
   - IN_SYNC = 1

Example

```
# get
input:  0/1 P_RECEIVESYNC ?
output: 0/1 P_RECEIVESYNC NO_SYNC
```

P_TRAFFIC

code: 124

```
# set
<module-index>/<port-index> P_TRAFFIC <on_off>

# get
<module-index>/<port-index> P_TRAFFIC ?
```

Description

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.

Note: From Release 57.1, if any of the specified packet sizes cannot fit into the packet generator, this command will return FAILED and not start the traffic. 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.

Actions

set, get

Parameters

1. `on_off`: *byte*, the traffic generation status of the port.

- `STOP = 0`
- `START = 1`

Example

```
# set
input:  0/1 P_TRAFFIC STOP
output: <OK>

# get
input:  0/1 P_TRAFFIC ?
output: 0/1 P_TRAFFIC STOP
```

P_TXDELAY

`code:  337`

```
# set
<module-index>/<port-index> P_TXDELAY <delay_val>

# get
<module-index>/<port-index> P_TXDELAY ?
```

Description

Sets a variable delay from a traffic start command received by the port until it starts transmitting. The delay is specified in multiples of 64 microseconds. Valid values are 0-31250 (0 to 2.000.000 microseconds).

Note: You must use `C_TRAFFIC` instead of `P_TRAFFIC` to start traffic for `P_TXDELAY` to have this effect.

Actions

set, get

Parameters

1. `delay_val`: *integer*, the delay specified in multiples of 64 microseconds.

Example

```
# set
input:  0/1 P_TXDELAY 1
output: <OK>

# get
input:  0/1 P_TXDELAY ?
output: 0/1 P_TXDELAY 1
```

P_TXENABLE

`code:  327`

```
# set
<module-index>/<port-index> P_TXENABLE <on_off>

# get
<module-index>/<port-index> P_TXENABLE ?
```

Description

Whether a port should enable its transmitter, or keep the outgoing link down.

Actions

set, get

Parameters

1. `on_off`: *byte*, the port's transmitter status
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 P_TXENABLE OFF
output: <OK>

# get
input:  0/1 P_TXENABLE ?
output: 0/1 P_TXENABLE OFF
```

P_TXPREPARE

`code:  336`

```
# set
<module-index>/<port-index> P_TXPREPARE
```

Description

Prepare port for transmission

Actions

set

Parameters

Example

```
# set
input:  0/1 P_TXPREPARE
output: <OK>
```

P_TXTIME

code: 330

```
# get
<module-index>/<port-index> P_TXTIME ?
```

Description

How long the port has been transmitting, the elapsed time since traffic was started.

Actions

get

Parameters

1. `microseconds`: *long integer*, how long the port has been transmitting, the elapsed time since traffic was started in microseconds

Example

```
# get
input:  0/1 P_TXTIME ?
output: 0/1 P_TXTIME 123456789123
```

P_XMITONE

code: 126

```
# set
<module-index>/<port-index> P_XMITONE <hex_data>
```

Description

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.

Actions

set

Parameters

1. hex_data: *hex list*, raw bytes of the packet in hex to transmit

Example

```
# set
input: 0/1 P_XMITONE␣
↪0x04f4bc19a2a104f4bc19a2a008004500003200000000 7fff12b70a0a0a010a0a0a0222232425262
output: <OK>
```

P_XMITONETIME

code: 331

```
# get
<module-index>/<port-index> P_XMITONETIME ?
```

Description

The time at which the latest packet was transmitted using the P_XMITONE command. The time reference is the same used by the time stamps of captured packets.

Actions

get

Parameters

1. `nanoseconds`: *long integer*, the time at which the latest packet was transmitted using the `P_XMITONE` command in nanoseconds

Example

```
# get
input:  0/1 P_XMITONETIME ?
output: 0/1 P_XMITONETIME 123456789123
```

Transmission Profile

P_RATEFRACTION

`code:  321`

```
# set
<module-index>/<port-index> P_RATEFRACTION <port_rate_ppm>

# get
<module-index>/<port-index> P_RATEFRACTION ?
```

Description

The port-level rate of the traffic transmitted for a port in sequential tx mode, expressed in millionths of the effective rate for the port. The bandwidth consumption includes the inter-frame gaps, and does not depend on the length of the packets for the streams.

Actions

set, get

Parameters

1. `port_rate_ppm`: *integer*, the port-level rate of the traffic transmitted for a port in sequential tx mode, expressed in millionths of the effective rate for the port

Example

```
# set
input:  0/1 P_RATEFRACTION 1
output: <OK>

# get
input:  0/1 P_RATEFRACTION ?
output: 0/1 P_RATEFRACTION 1
```

P_RATEL2BPS

code: 323

```
# set
<module-index>/<port-index> P_RATEL2BPS <port_rate_bps>

# get
<module-index>/<port-index> P_RATEL2BPS ?
```

Description

The port-level rate of the traffic transmitted for a port in sequential tx mode, 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.

Actions

set, get

Parameters

1. `port_rate_bps`: *long integer*, the port-level rate of the traffic transmitted for a port in sequential tx mode, expressed in units of bits per-second at layer-2, thus including the Ethernet header but excluding the inter-frame gap

Example

```
# set
input:  0/1 P_RATEL2BPS 1
output: <OK>

# get
input:  0/1 P_RATEL2BPS ?
output: 0/1 P_RATEL2BPS 1
```

P_RATEPPS

code: 322

```
# set
<module-index>/<port-index> P_RATEPPS <port_rate_pps>

# get
<module-index>/<port-index> P_RATEPPS ?
```

Description

The port-level rate of the traffic transmitted for a port in sequential tx mode, expressed in packets per second. The bandwidth consumption is heavily dependent on the length of the packets generated for the streams, and also on the inter- frame gap for the port.

Actions

set, get

Parameters

1. `port_rate_pps`: *integer*, the port-level rate of the traffic transmitted for a port in sequential tx mode, expressed in packets per second

Example

```
# set
input:  0/1 P_RATEPPS 1
output: <OK>

# get
input:  0/1 P_RATEPPS ?
output: 0/1 P_RATEPPS 1
```

P_TPLDMODE

`code:  350`

```
# set
<module-index>/<port-index> P_TPLDMODE <mode>

# get
<module-index>/<port-index> P_TPLDMODE ?
```

Description

Sets the size of the Xena Test Payload (TPLD) used to track streams, perform latency measurements etc.

- Default is "Normal", which is a 20 byte TPLD.
- "Micro" is a condensed version, which is useful when generating very small packets with relatively long headers (like IPv6). It has the following characteristics compared to the "normal" TPLD. * Only 6 byte long. * Less accurate mechanism to separate Xena-generated packets from other packets is the network - it is recommended not to have too much other traffic going into the receive Xena port, when micro TPLD is used. * No sequence checking (packet loss or packet misordering). The number of received packets for each stream can still be compared to the number of transmitted packets to detect packet loss once traffic has been stopped.

When the TPLDMODE is changed, it will affect ALL streams on the port.

Actions

set, get

Parameters

1. `mode`: *byte*, the Test Payload mode of the port.

- `NORMAL = 0`
- `MICRO = 1`

Example

```
# set
input:  0/1 P_TPLDMODE NORMAL
output: <OK>

# get
input:  0/1 P_TPLDMODE ?
output: 0/1 P_TPLDMODE NORMAL
```

P_TXBURSTPERIOD

`code:  377`

```
# set
<module-index>/<port-index> P_TXBURSTPERIOD <burst_period>

# get
<module-index>/<port-index> P_TXBURSTPERIOD ?
```

Description

In Burst TX mode this command defines the time from the start of one sequence of bursts (from a number of streams) to the start of next sequence of bursts. NB: Only used when Port TX Mode is "BURST".

Actions

set, get

Parameters

1. `burst_period`: *long integer*, the duration in microseconds from the start of one sequence of bursts (from a number of streams) to the start of next sequence of bursts in Burst TX mode

Example

```
# set
input:  0/1 P_TXBURSTPERIOD 1
output: <OK>

# get
input:  0/1 P_TXBURSTPERIOD ?
output: 0/1 P_TXBURSTPERIOD 1
```

P_TXMODE

`code:  320`

```
# set
<module-index>/<port-index> P_TXMODE <mode>

# get
<module-index>/<port-index> P_TXMODE ?
```

Description

The scheduling mode for outgoing traffic from the port, specifying how multiple logical streams are merged onto one physical port. There are four primary modes:

Normal Interleaved: The streams are treated independently, and are merged into a combined traffic pattern for the port, which honors each stream's ideal packet placements as well as possible. This is the default mode.

Strict Uniform: This is a slight variation of normal interleaved scheduling, which emphasizes strict uniformity of the inter-packet-gaps as more important than hitting the stream rates absolutely precisely.

Sequential: Each stream in turn contribute one or more packets, before continuing to the next stream, in a cyclical pattern. The count of packets for each stream is obtained from the

PS_PACKETLIMIT command value for the stream. The individual rates for each stream are ignored, and instead the overall rate is determined at the port-level. This in turn determines the rates for each stream, taking into account their packet lengths and counts. The maximum number of packets in a cycle (i.e. the sum of PS_PACKETLIMIT for all enabled streams) is 500. If the packet number is larger than 500, will be returned when attempting to start the traffic (P_TRAFFIC ON).

Burst*: When this mode is selected, frames from the streams on a port are sent as bursts as depicted below: The Burst Period is defined in the P_TXBURSTPERIOD command. For the individual streams the number of packets in a burst is defined by the PS_BURST command, while the Inter Packet Gap and the Inter Burst Gap are defined by the PS_BURSTGAP command.

Actions

set, get

Parameters

1. mode: *byte*, the scheduling mode for outgoing traffic from the port, containing the loopback mode for the port:

- NORMAL = 0, interleaved packet scheduling.
- STRICTUNIFORM = 1, strict uniform mode.
- SEQUENTIAL = 2, sequential packet scheduling.
- BURST = 3, burst mode.

Example

```
# set
input:  0/1 P_TXMODE NORMAL
output: <OK>

# get
input:  0/1 P_TXMODE ?
output: 0/1 P_TXMODE NORMAL
```

P_TXPACKETLIMIT

code: 352

```
# set
<module-index>/<port-index> P_TXPACKETLIMIT <packet_count_limit>

# get
<module-index>/<port-index> P_TXPACKETLIMIT ?
```

Description

The number of packets that will be transmitted from a port when traffic is started on the port. A value of 0 or -1 makes the port transmit continuously. Traffic from the streams on the port can however also be set to stop after transmitting a number of packets.

Actions

set, get

Parameters

1. `packet_count_limit`: *integer*, the number of packets that will be transmitted from the port when traffic is started on the port

Example

```
# set
input:  0/1 P_TXPACKETLIMIT 1
output: <OK>

# get
input:  0/1 P_TXPACKETLIMIT ?
output: 0/1 P_TXPACKETLIMIT 1
```

P_TXTIMELIMIT

code: 329

```
# set
<module-index>/<port-index> P_TXTIMELIMIT <microseconds>

# get
<module-index>/<port-index> P_TXTIMELIMIT ?
```

Description

A port-level time-limit on how long it keeps transmitting when started. After the elapsed time traffic must be stopped and restarted. This complements the stream-level PS_PACKETLIMIT function.

Actions

set, get

Parameters

1. `microseconds`: *long integer*, the port-level time-limit on how long it keeps transmitting when started in microseconds. Maximum can be 2^63

Example

```
# set
input:  0/1 P_TXTIMELIMIT 1
output: <OK>

# get
input:  0/1 P_TXTIMELIMIT ?
output: 0/1 P_TXTIMELIMIT 1
```

Loopback Mode

P_LOOPBACK

code: 122

```
# set
<module-index>/<port-index> P_LOOPBACK <mode>

# get
<module-index>/<port-index> P_LOOPBACK ?
```

Description

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

1. Off: Traffic flows naturally out of the port (illustrated in Fig. 3.2)
2. External RX-to-TX loopback (illustrated in Fig. 3.4), where the received packets are retransmitted immediately. The packets are still processed by the receive logic, and can be captured and analyzed.

- L1 RX-to-TX: Any received packet is bounced back through TX
- L2 RX-to-TX: Same as L1 RX-to-TX yet it also swaps SRC MAC address with DST MAC address
- L3 RX-to-TX: Same as L1 RX-to-TX yet it also swaps SRC IP address with DST IP address

3. Internal TX-to-RX loopback (illustrated in Fig. 3.3), where the transmitted packets are received directly by the port itself. This is mainly useful for testing the generated traffic patterns before actual use.

- TX(on)-to-RX: Packet goes out of TX but also internally direct to RX
- TX(off)-to-TX: Packet goes directly to RX (No link sync needed)

4. Port-to-port: Any received packet goes out through the neighbor port (illustrated in Fig. 3.5)

Actions

set, get

Parameters

1. `mode`: *byte*, the loop back mode of the port

- `NONE = 0`
- `L1RX2TX = 1`
- `L2RX2TX = 2`
- `L3RX2TX = 3`
- `TXON2RX = 4`
- `TXOFF2RX = 5`
- `PORT2PORT = 6`

Example

```
# set
input:  0/1 P_LOOPBACK NONE
output: <OK>

# get
input:  0/1 P_LOOPBACK ?
output: 0/1 P_LOOPBACK NONE
```

Illustrations

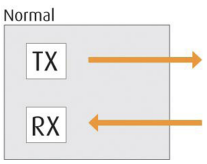


Fig. 3.2: Loopback Mode - Normal

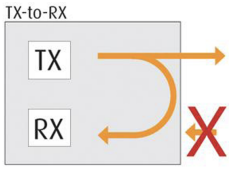


Fig. 3.3: Loopback Mode - TX-to-RX

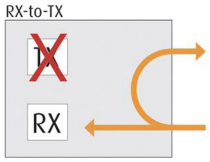


Fig. 3.4: Loopback Mode - RX-to-TX

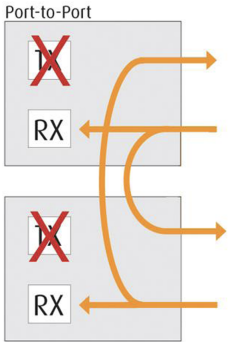


Fig. 3.5: Loopback Mode - Port-to-Port

Latency Mode

P_LATENCYMODE

code: 128

```
# set
<module-index>/<port-index> P_LATENCYMODE <mode>

# get
<module-index>/<port-index> P_LATENCYMODE ?
```

Description

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 four 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 occur.
- First-bit-out to first-bit-in, which adds the time taken to transmit the packet itself, and 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 occur.

Actions

set, get

Parameters

1. mode: *byte*, the latency measurement mode of the port

- LAST2LAST = 0
- FIRST2LAST = 1
- LAST2FIRST = 2
- FIRST2FIRST = 3

Example

```
# set
input:  0/1 P_LATENCYMODE LAST2LAST
output: <OK>

# get
input:  0/1 P_LATENCYMODE ?
output: 0/1 P_LATENCYMODE LAST2LAST
```

P_LATENCYOFFSET

code: 127

```
# set
<module-index>/<port-index> P_LATENCYOFFSET <offset>

# get
<module-index>/<port-index> P_LATENCYOFFSET ?
```

Description

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 command.

Actions

set, get

Parameters

1. `offset`: *integer*, the port latency offset value in nanoseconds

Example

```
# set
input:  0/1 P_LATENCYOFFSET 1
output: <OK>

# get
input:  0/1 P_LATENCYOFFSET ?
output: 0/1 P_LATENCYOFFSET 1
```

MAC Control

P_AUTOTRAIN

code: 129

```
# set
<module-index>/<port-index> P_AUTOTRAIN <interval>

# get
<module-index>/<port-index> P_AUTOTRAIN ?
```

Description

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.

Actions

set, get

Parameters

1. `interval`: *integer*, the interval between sending out training packets of the port

Example

```
# set
input:  0/1 P_AUTOTRAIN 1
output: <OK>

# get
input:  0/1 P_AUTOTRAIN ?
output: 0/1 P_AUTOTRAIN 1
```

P_CHECKSUM

`code:  302`

```
# set
<module-index>/<port-index> P_CHECKSUM <offset>

# get
<module-index>/<port-index> P_CHECKSUM ?
```

Description

Controls 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).

Actions

set, get

Parameters

1. `offset`: *short integer*, the offset in the packet where the calculation of the extra checksum is started from

Example

```
# set
input:  0/1 P_CHECKSUM 1
output: <OK>

# get
input:  0/1 P_CHECKSUM ?
output: 0/1 P_CHECKSUM 1
```

P_GAPMONITOR

code: 301

```
# set
<module-index>/<port-index> P_GAPMONITOR <start> <stop>

# get
<module-index>/<port-index> P_GAPMONITOR ?
```

Description

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.

Actions

set, get

Parameters

1. `start`: *integer*, the maximum allowed gap between packets, in microseconds. (0 to 134.000 microseconds) 0 = disable gap monitor
2. `stop`: *integer*, the minimum number of good packets required. (0 to 1024 packets) 0 = disable gap monitor

Example

```
# set
input:  0/1 P_GAPMONITOR 1 1
output: <OK>

# get
input:  0/1 P_GAPMONITOR ?
output: 0/1 P_GAPMONITOR 1 1
```

P_INTERFRAMEGAP

`code:  114`

```
# set
<module-index>/<port-index> P_INTERFRAMEGAP <min_byte_count>

# get
<module-index>/<port-index> P_INTERFRAMEGAP ?
```

Description

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

Actions

set, get

Parameters

1. `min_byte_count`: *integer*, the minimum gap between packets in the traffic generated for a port. The gap includes the Ethernet preamble.

Example

```
# set
input:  0/1 P_INTERFRAMEGAP 1
output: <OK>

# get
input:  0/1 P_INTERFRAMEGAP ?
output: 0/1 P_INTERFRAMEGAP 1
```

P_MACADDRESS

`code:  116`

```
# set
<module-index>/<port-index> P_MACADDRESS <mac_address>

# get
<module-index>/<port-index> P_MACADDRESS ?
```

Description

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.

Actions

set, get

Parameters

1. mac_address: *hex6*, the MAC address of the port

Example

```
# set
input:  0/1 P_MACADDRESS 0x1234FFFFE1E2
output: <OK>

# get
input:  0/1 P_MACADDRESS ?
output: 0/1 P_MACADDRESS 0x1234FFFFE1E2
```

P_PAUSE

code: 120

```
# set
<module-index>/<port-index> P_PAUSE <on_off>

# get
<module-index>/<port-index> P_PAUSE ?
```

Description

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

Actions

set, get

Parameters

1. on_off: *byte*, the status of whether the port responds to incoming Ethernet PAUSE frames by holding back outgoing traffic.

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P_PAUSE OFF
output: <OK>

# get
input:  0/1 P_PAUSE ?
output: 0/1 P_PAUSE OFF
```

PFC

P_PFCENABLE

code: 373

```
# set
<module-index>/<port-index> P_PFCENABLE <cos_0> <cos_1> <cos_2> <cos_3>
 ↪ <cos_4> <cos_5> <cos_6> <cos_7>

# get
<module-index>/<port-index> P_PFCENABLE ?
```

Description

This setting control whether a port responds to incoming Ethernet Priority Flow Control (PFC) frames, by holding back outgoing traffic for that priority.

Actions

set, get

Parameters

1. `cos_0`: *byte*, whether PFC response is enabled for CoS 0
   - `OFF = 0`
   - `ON = 1`
2. `cos_1`: *byte*, whether PFC response is enabled for CoS 1
   - `OFF = 0`
   - `ON = 1`

3. cos_2: *byte*, whether PFC response is enabled for CoS 2
   - OFF = 0
   - ON = 1
4. cos_3: *byte*, whether PFC response is enabled for CoS 3
   - OFF = 0
   - ON = 1
5. cos_4: *byte*, whether PFC response is enabled for CoS 4
   - OFF = 0
   - ON = 1
6. cos_5: *byte*, whether PFC response is enabled for CoS 5
   - OFF = 0
   - ON = 1
7. cos_6: *byte*, whether PFC response is enabled for CoS 6
   - OFF = 0
   - ON = 1
8. cos_7: *byte*, whether PFC response is enabled for CoS 7
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_PFCENABLE OFF OFF OFF OFF OFF OFF OFF OFF
output: <OK>

# get
input:  0/1 P_PFCENABLE ?
output: 0/1 P_PFCENABLE OFF OFF OFF OFF OFF OFF OFF OFF
```

Payload

P_MAXHEADERLENGTH

code: 328

```
# set
<module-index>/<port-index> P_MAXHEADERLENGTH <max_header_length>

# get
<module-index>/<port-index> P_MAXHEADERLENGTH ?
```

Description

The maximum number of header content bytes that can be freely specified for each generated stream. The remaining payload bytes of the packet are auto- generated.The default is 128 bytes. When a larger number is select there is a corresponding proportional reduction in the number of stream definitions that are available for the port. Possible values: 128 (default), 256, 512, 1024, 2048.

Actions

set, get

Parameters

1. `max_header_length`: *integer*, the maximum number of header content bytes that can be freely specified for each generated stream on the port

Example

```
# set
input:  0/1 P_MAXHEADERLENGTH 1
output: <OK>

# get
input:  0/1 P_MAXHEADERLENGTH ?
output: 0/1 P_MAXHEADERLENGTH 1
```

P_MIXLENGTH

code: 305

```
# set
<module-index>/<port-index> P_MIXLENGTH [<position_index>] <frame_size>

# get
<module-index>/<port-index> P_MIXLENGTH [<position_index>] ?
```

Description

Allows inspecting the frame sizes defined for each position of the P_MIXWEIGHTS command. By default, the 16 frame sizes are: 56 (not valid for 40G/100G), 60, 64, 70, 78, 92, 256, 496, 512, 570, 576, 594, 1438, 1518, 9216, and 16360. In addition to inspecting these sizes one by one, it also allows changing frame size for positions 0, 1, 14 and 15 (default values 56, 60, 9216 and 16360). Supported by the following modules: Thor-400G-7S-1P, Thor-100G-5S-4P and Loki-100G-5S-2P.

Note: This command requires release 84 or higher.

Actions

set, get

Parameters

1. `frame_size`: *integer*, the frame size for the position.

Example

```
# set
input:  0/1 P_MIXLENGTH [0] 1
output: <OK>

# get
input:  0/1 P_MIXLENGTH [0] ?
output: 0/1 P_MIXLENGTH [0] 1
```

P_MIXWEIGHTS

code: 192

```
# set
<module-index>/<port-index> P_MIXWEIGHTS <weight_56_bytes> <weight_60_
↪bytes> <weight_64_bytes> <weight_70_bytes> <weight_78_bytes> <weight_
↪92_bytes> <weight_256_bytes> <weight_496_bytes> <weight_512_bytes>
↪<weight_570_bytes> <weight_576_bytes> <weight_594_bytes> <weight_
↪1438_bytes> <weight_1518_bytes> <weight_9216_bytes> <weight_16360_
↪bytes>

# get
<module-index>/<port-index> P_MIXWEIGHTS ?
```

Description

Allow changing the distribution of the MIX packet length by specifying the percentage of each of the 16 possible frame sizes used in the MIX. The sum of the percentage values specified must be 100. The command will affect the mix- distribution for all streams on the port. The possible 16 frame sizes are: 56 (not valid for 40G/100G), 60, 64, 70, 78, 92, 256, 496, 512, 570, 576, 594, 1438, 1518, 9216, and 16360.

Note: This command requires Xena server version 375 or higher.

Actions

set, get

Parameters

1. `weight_56_bytes`: *integer*, specifying the percentage of 56-byte frame sizes
2. `weight_60_bytes`: *integer*, specifying the percentage of 60-byte frame sizes
3. `weight_64_bytes`: *integer*, specifying the percentage of 64-byte frame sizes
4. `weight_70_bytes`: *integer*, specifying the percentage of 70-byte frame sizes
5. `weight_78_bytes`: *integer*, specifying the percentage of 78-byte frame sizes
6. `weight_92_bytes`: *integer*, specifying the percentage of 92-byte frame sizes
7. `weight_256_bytes`: *integer*, specifying the percentage of 256-byte frame sizes
8. `weight_496_bytes`: *integer*, specifying the percentage of 496-byte frame sizes

9. `weight_512_bytes`: *integer*, specifying the percentage of 512-byte frame sizes
10. `weight_570_bytes`: *integer*, specifying the percentage of 570-byte frame sizes
11. `weight_576_bytes`: *integer*, specifying the percentage of 576-byte frame sizes
12. `weight_594_bytes`: *integer*, specifying the percentage of 594-byte frame sizes
13. `weight_1438_bytes`: *integer*, specifying the percentage of 1438-byte frame sizes
14. `weight_1518_bytes`: *integer*, specifying the percentage of 1518-byte frame sizes
15. `weight_9216_bytes`: *integer*, specifying the percentage of 9216-byte frame sizes
16. `weight_16360_bytes`: *integer*, specifying the percentage of 16360-byte frame sizes

Example

```
# set
input:  0/1 P_MIXWEIGHTS 0 0 0 0 57 3 5 1 2 5 1 4 4 18 0 0
output: <OK>

# get
input:  0/1 P_MIXWEIGHTS ?
output: 0/1 P_MIXWEIGHTS 0 0 0 0 57 3 5 1 2 5 1 4 4 18 0 0
```

P_PAYLOADMODE

code: 324

```
# set
<module-index>/<port-index> P_PAYLOADMODE <mode>

# get
<module-index>/<port-index> P_PAYLOADMODE ?
```

Description

Set this command to configure the port to use different payload modes, i.e. normal, extend payload, and custom payload field, for ALL streams on this port. The extended payload feature allows the definition of a much larger (up to MTU) payload buffer for each stream. The custom payload field feature allows you to define a sequence of custom data fields for each stream. The data fields will then be used in a round robin fashion when packets are sent based on the stream definition.

Actions

set, get

Parameters

1. `mode`: *byte*, the port's payload mode for **ALL** streams on this port.

- `NORMAL = 0`, normal.
- `EXTPL = 1`, extended payload.
- `CDF = 2`, custom payload field.

Example

```
# set
input:  0/1 P_PAYLOADMODE NORMAL
output: <OK>

# get
input:  0/1 P_PAYLOADMODE ?
output: 0/1 P_PAYLOADMODE NORMAL
```

P_RANDOMSEED

`code:  121`

```
# set
<module-index>/<port-index> P_RANDOMSEED <seed>

# get
<module-index>/<port-index> P_RANDOMSEED ?
```

Description

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.

Actions

set, get

Parameters

1. `seed`: *integer*, the seed value for the port

Example

```
# set
input:  0/1 P_RANDOMSEED 1
output: <OK>

# get
input:  0/1 P_RANDOMSEED ?
output: 0/1 P_RANDOMSEED 1
```

IPv4

P_ARPREPLY

`code:  118`

```
# set
<module-index>/<port-index> P_ARPREPLY <on_off>

# get
<module-index>/<port-index> P_ARPREPLY ?
```

Description

Whether the port replies to ARP requests. 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/NDP reply generation is independent of whether traffic and capture is on for the port.

Actions

set, get

Parameters

1. `on_off`: *byte*, whether the port replies to ARP requests

- `OFF = 0`
- `ON = 1`

Example

```
# set
input:  0/1 P_ARPREPLY OFF
output: <OK>

# get
input:  0/1 P_ARPREPLY ?
output: 0/1 P_ARPREPLY OFF
```

P_ARPRXTABLE

`code:  308`

```
# set
<module-index>/<port-index> P_ARPRXTABLE <entries>

# get
<module-index>/<port-index> P_ARPRXTABLE ?
```

Description

Port ARP table used to reply to incoming ARP requests.

Actions

set, get

Parameters

1. `entries`: *hex13 list*, each `hex13` value corresponds to an entry in the ARP table, which contains the following parameters

- `ipv4_address`: *hex4*, IP address to match to the Target IP address in the ARP requests
- `prefix`: *hex2*, The prefix used for address matching
- `patched_mac`: *hex*, Whether the target MAC address will be patched with the part of the IP address that is not masked by the prefix
- `mac_address`: *hex6*, The target MAC address to return in the ARP reply

Example

```
# set
input:  0/1 P_ARPRXTABLE␣
↪0x0A0A0A0A00180111223344556 60B0B0B0B00180199887766 5544
output: <OK>

# get
input:  0/1 P_ARPRXTABLE ?
output: 0/1 P_ARPRXTABLE␣
↪0x0A0A0A0A00180111223344556 60B0B0B0B00180199887766 5544


0x0A0A0A0A00180111223344556 60B0B0B0B00180199887766 5544 corresponds to␣
↪two entries in the ARP RX table:

===========   =======  ==================   ==============
IP Address    Prefix   MAC Address          Patch MAC
===========   =======  ==================   ==============
10.10.10.10   24       11 22 33 44 55 66    Yes
11.11.11.11   24       11 22 33 44 55 66    Yes
===========   =======  ==================   ==============
```

P_IPADDRESS

code: 117

```
# set
<module-index>/<port-index> P_IPADDRESS <ipv4_address> <subnet_mask>
 ↪<gateway> <wild>

# get
<module-index>/<port-index> P_IPADDRESS ?
```

Description

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.

Actions

set, get

Parameters

1. `ipv4_address`: *address*, the IPv4 address of the port
2. `subnet_mask`: *address*, the subnet mask of the local network segment for the port
3. `gateway`: *address*, he gateway of the local network segment for the port
4. `wild`: *address*, wildcards used for ARP and PING replies, and each byte must be 255 (0xFF) or 0 (0x00)

Example

```
# set
input:  0/1 P_IPADDRESS 192.168.1.100 255.255.255.0 192.168.1.1 255.
 ↪255.255.255
output: <OK>

# get
input:  0/1 P_IPADDRESS ?
output: 0/1 P_IPADDRESS 192.168.1.100 255.255.255.0 192.168.1.1 255.
 ↪255.255.255
```

P_PINGREPLY

code: 119

```
# set
<module-index>/<port-index> P_PINGREPLY <on_off>

# get
<module-index>/<port-index> P_PINGREPLY ?
```

Description

Whether the port replies to IPv4/IPv6 PING. The port can reply to incoming IPv4/IPv6 PING requests to the IP address specified for the port. IPv4/IPv6 PING reply generation is independent of whether traffic and capture is on for the port.

Actions

set, get

Parameters

1. on_off: *byte*, whether the port replies to IPv4/IPv6 PING requests
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_PINGREPLY OFF
output: <OK>

# get
input:  0/1 P_PINGREPLY ?
output: 0/1 P_PINGREPLY OFF
```

IPv6

P_ARPV6REPLY

code: 333

```
# set
<module-index>/<port-index> P_ARPV6REPLY <on_off>

# get
<module-index>/<port-index> P_ARPV6REPLY ?
```

Description

Whether the port generates replies using the IPv6 Network Discovery Protocol. The port can reply to incoming NDP Neighbor Solicitations by mapping the IPv6 address specified for the port to the MAC address specified for the port. NDP reply generation is independent of whether traffic and capture is on for the port.

Actions

set, get

Parameters

1. on_off: *byte*, whether the port replies to NDP Neighbor Solicitations.
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_ARPV6REPLY OFF
output: <OK>

# get
input:  0/1 P_ARPV6REPLY ?
output: 0/1 P_ARPV6REPLY OFF
```

P_IPV6ADDRESS

code: 332

```
# set
<module-index>/<port-index> P_IPV6ADDRESS <ipv6_address> <gateway>
 ↪<subnet_prefix> <wildcard_prefix>

# get
<module-index>/<port-index> P_IPV6ADDRESS ?
```

Description

An IPv6 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 NDP and PINGv6 protocols.

Actions

set, get

Parameters

1. `ipv6_address`: *string*, the IPv6 address of the port
2. `gateway`: *string*, the gateway of the local network segment for the port
3. `subnet_prefix`: *short integer*, the subnet prefix of the local network segment for the port
4. `wildcard_prefix`: *short integer*, a prefix that makes the port replies to NDP/PING for the masked addresses, valid value 0-255

Example

```
# set
input:  0/1 P_IPV6ADDRESS ::1 192.168.1.1 1 1
output: <OK>

# get
input:  0/1 P_IPV6ADDRESS ?
output: 0/1 P_IPV6ADDRESS ::1 192.168.1.1 1 1
```

P_NDPRXTABLE

code: 309

```
# set
<module-index>/<port-index> P_NDPRXTABLE <entries>

# get
<module-index>/<port-index> P_NDPRXTABLE ?
```

Description

Port NDP table used to reply to incoming NDP Neighbor Solicitation.

Actions

set, get

Parameters

1. `entries`: *hex25 list*, each `hex25` value corresponds to an entry in the ARP table, which contains the following parameters

- `ipv6_address`: *hex16*, IP address to match to the Target IP address in the NDP Neighbor Solicitation
- `prefix`: *hex2*, The prefix used for address matching
- `patched_mac`: *hex*, Whether the target MAC address will be patched with the part of the IP address that is not masked by the prefix
- `mac_address`: *hex6*, The target MAC address to return in the NDP Neighbor Advertisement

Example

```
# set
input: 0/1 P_NDPRXTABLE␣
↪0x00000000000000000000000000000123008001112233445566
output: <OK>

# get
input: 0/1 P_NDPRXTABLE ?
output: 0/1 P_NDPRXTABLE␣
↪0x00000000000000000000000000000123008001112233445566
```

(continues on next page)

(continued from previous page)

```
0x00000000000000000000000000001230080011122334455 66 corresponds to␣
↪one entry in the NDP RX table:

===========   =======  ==================   ==============
IP Address    Prefix   MAC Address          Patch MAC
===========   =======  ==================   ==============
::123         128       11 22 33 44 55 66   Yes
===========   =======  ==================   ==============
```

P_PINGV6REPLY

code: 334

```
# set
<module-index>/<port-index> P_PINGV6REPLY <on_off>

# get
<module-index>/<port-index> P_PINGV6REPLY ?
```

Description

Whether the port generates PINGv6 replies using the ICMP protocol received over IPv6. The port can reply to incoming PINGv6 requests to the IPv6 address specified for the port. PINGv6 reply generation is independent of whether traffic and capture is on for the port.

Actions

set, get

Parameters

1. on_off: *byte*, whether the port replies to incoming PINGv6.
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_PINGV6REPLY OFF
output: <OK>

# get
input:  0/1 P_PINGV6REPLY ?
output: 0/1 P_PINGV6REPLY OFF
```

Multicast

P_MCSRCLIST

code: 313

```
# set
<module-index>/<port-index> P_MCSRCLIST <ipv4_addresses>

# get
<module-index>/<port-index> P_MCSRCLIST ?
```

Description

Multicast source list of the port. Only valid if the IGMP protocol version is IGMPv3 set by P_MULTICASTEXT.

Actions

set, get

Parameters

1. `ipv4_addresses`: *address list*, the multicast source list of the port

Example

```
# set
input:  0/1 P_MCSRCLIST 192.168.1.100
output: <OK>

# get
input:  0/1 P_MCSRCLIST ?
output: 0/1 P_MCSRCLIST 192.168.1.100
```

P_MULTICAST

code: 311

```
# set
<module-index>/<port-index> P_MULTICAST <ipv4_multicast_addresses>
↪<operation> <second_count>

# get
<module-index>/<port-index> P_MULTICAST ?
```

Description

A multicast mode for a port. Ports can use the IGMPv2 protocol to join or leave multicast groups, either on an on-off basis or repeatedly.

Actions

set, get

Parameters

1. `ipv4_multicast_addresses`: *address list*, a multicast group address to join or leave
2. `operation`: *byte*, the operation
   - `OFF = 0`
   - `ON = 1`
   - `JOIN = 2`
   - `LEAVE = 3`
3. `second_count`: *short integer*, the interval between repeated joins in seconds.

Example

```
# set
input:  0/1 P_MULTICAST 192.168.1.100 OFF 1
output: <OK>

# get
input:  0/1 P_MULTICAST ?
output: 0/1 P_MULTICAST 192.168.1.100 OFF 1
```

P_MULTICASTEXT

code: 312

```
# set
<module-index>/<port-index> P_MULTICASTEXT <ipv4_multicast_addresses>
↪<operation> <second_count> <igmp_version>

# get
<module-index>/<port-index> P_MULTICASTEXT ?
```

Description

A multicast mode for a port. Ports can use the IGMPv2/IGMPv3 protocol to join or leave multicast groups, either on an on-off basis or repeatedly. ** Requires software release 83.2 or higher

Actions

set, get

Parameters

1. `ipv4_multicast_addresses`: *address list*, a multicast group address to join or leave
2. `operation`: *byte*, the operation

- `OFF = 0`
- `ON = 1`
- `JOIN = 2`
- `LEAVE = 3`
- `INCLUDE = 4`

- EXCLUDE = 5
- LEAVE_TO_ALL = 6
- GENERAL_QUERY = 7
- GROUP_QUERY = 8

3. second_count: *short integer*, the interval between repeated joins in seconds.
4. igmp_version: *byte*, IGMP version

- IGMPV2 = 0
- IGMPV3 = 1

Example

```
# set
input:  0/1 P_MULTICASTEXT 192.168.1.100 OFF 1 IGMPV2
output: <OK>

# get
input:  0/1 P_MULTICASTEXT ?
output: 0/1 P_MULTICASTEXT 192.168.1.100 OFF 1 IGMPV2
```

P_MULTICASTHDR

code: 314

```
# set
<module-index>/<port-index> P_MULTICASTHDR <header_count> <header_
↪format> <tag> <pcp> <dei>

# get
<module-index>/<port-index> P_MULTICASTHDR ?
```

Description

Allows addition of a VLAN tag to IGMPv2 and IGPMv3 packets. This command requires software release 83.2 or higher.

Actions

set, get

Parameters

1. `header_count`: *short integer*, number of additional headers. Currently only 0 or 1 supported
2. `header_format`: *byte*, indicates the header format
   - `NOHDR = 0`
   - `VLAN = 1`
3. `tag`: *integer*, VLAN tag (VID)
4. `pcp`: *short integer*, VLAN Priority code point
5. `dei`: *byte*, drop-eligible indicator
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 P_MULTICASTHDR 1 NOHDR 1 1 OFF
output: <OK>

# get
input:  0/1 P_MULTICASTHDR ?
output: 0/1 P_MULTICASTHDR 1 NOHDR 1 1 OFF
```

Histogram

Port histogram commands that deal with configuration of data collection and retrieval of samples from a port.

The histogram command names all have the form `PD_<xxx>` and require both a module index id and a port index id, as well as a sub-index identifying a particular histogram.

A histogram 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 histogram is actively collecting samples its parameters cannot be changed.

PD_CONFIG

code: 131

```
# get
<module-index>/<port-index> PD_CONFIG [<dataset_index>] ?
```

Description

Return all relevant values for dataset

Actions

get

Parameters

Example

```
# get
input:  0/1 PD_CONFIG [0] ?
```

PD_CREATE

code: 141

```
# set
<module-index>/<port-index> PD_CREATE [<dataset_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PD_CREATE [0]
output: <OK>
```

PD_DELETE

code: 142

```
# set
<module-index>/<port-index> PD_DELETE [<dataset_index>]
```

Description

Delete an existing histogram definition.

Actions

set

Parameters

Example

```
# set
input:  0/1 PD_DELETE [0]
output: <OK>
```

PD_ENABLE

code: 143

```
# set
<module-index>/<port-index> PD_ENABLE [<dataset_index>] <on_off>

# get
<module-index>/<port-index> PD_ENABLE [<dataset_index>] ?
```

Description

Whether a histogram is currently active on a port. When turned on, all the bucket counts are cleared to zero. Subsequently each packet matching the histogram source criteria is counted into one of the buckets. While a histogram is enabled its parameters cannot be changed.

Actions

set, get

Parameters

1. on_off: *byte*, whether the histogram is enabled

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 PD_ENABLE [0] OFF
output: <OK>

# get
input:  0/1 PD_ENABLE [0] ?
output: 0/1 PD_ENABLE [0] OFF
```

PD_INDICES

code: 140

```
# set
<module-index>/<port-index> PD_INDICES <histogram_indices>

# get
<module-index>/<port-index> PD_INDICES ?
```

Description

Obtain or configure histogram indices for each of N histograms.

Actions

set, get

Parameters

1. `histogram_indices`: *integer list*, histogram indices

Example

```
# set
input:  0/1 PD_INDICES 0 1
output: <OK>

# get
input:  0/1 PD_INDICES ?
output: 0/1 PD_INDICES 0 1
```

PD_RANGE

code: 145

```
# set
<module-index>/<port-index> PD_RANGE [<dataset_index>] <start> <step>
↪<bucket_count>

# get
<module-index>/<port-index> PD_RANGE [<dataset_index>] ?
```

Description

The bucket ranges used for classifying the packets counted by a histogram 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 do not 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.

Actions

set, get

Parameters

1. `start`: *integer*, first value going into the second bucket
2. `step`: *integer*, the span of each middle bucket: (1) 1,2,4,8,16,32,64,128,256,512 (bytes, non-latency histograms). (2) 16,32,64,128,...,1048576,2097152 (nanoseconds, latency histograms)
3. `bucket_count`: *integer*, the total number of buckets

Example

```
# set
input:  0/1 PD_RANGE [0] 1 1 1
output: <OK>

# get
input:  0/1 PD_RANGE [0] ?
output: 0/1 PD_RANGE [0] 1 1 1
```

PD_SAMPLES

`code:  146`

```
# get
<module-index>/<port-index> PD_SAMPLES [<dataset_index>] ?
```

Description

The current set of counts collected by a histogram 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.

Actions

get

Parameters

1. `packet_counts`: *long integer list*, the number of packets counted for each bucket

Example

```
# get
input:  0/1 PD_SAMPLES [0] ?
output: 0/1 PD_SAMPLES [0] 123456789123 123456789124
```

PD_SOURCE

`code:  144`

```
# set
<module-index>/<port-index> PD_SOURCE [<dataset_index>] <source_type>
↪<which_packets> <identity>

# get
<module-index>/<port-index> PD_SOURCE [<dataset_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `source_type`: *byte*, what is counted and for which packets
   - `TXIFG = 0`
   - `TXLEN = 1`
   - `RXIFG = 2`
   - `RXLEN = 3`
   - `RXLAT = 4`
   - `RXJIT = 5`
2. `which_packets`: *byte*, a further detail on which packets to count
   - `ALL = 0`
   - `TPLD = 1`
   - `FILTER = 2`
3. `identity`: *integer*, test payload id or filter id for the wanted packets

Example

```
# set
input:  0/1 PD_SOURCE [0] TXIFG ALL 1
output: <OK>

# get
input:  0/1 PD_SOURCE [0] ?
output: 0/1 PD_SOURCE [0] TXIFG ALL 1
```

Filter

Filter Configuration

Port filter commands that deal with configuration of the filters on the received traffic of a port.

The filter command names all have the form `PF_<xxx>`, and require both a module index id and a port index id, as well as a sub-index identifying a particular filter.

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.

PF_COMMENT

code: 215

```
# set
<module-index>/<port-index> PF_COMMENT [<filter_index>] <comment>

# get
<module-index>/<port-index> PF_COMMENT [<filter_index>] ?
```

Description

The description of a filter.

Actions

set, get

Parameters

1. comment: *string*, the description of the filter.

Example

```
# set
input:  0/1 PF_COMMENT [0] 'this is a comment'
output: <OK>

# get
input:  0/1 PF_COMMENT [0] ?
output: 0/1 PF_COMMENT [0] 'this is a comment'
```

PF_CONDITION

code: 216

```
# set
<module-index>/<port-index> PF_CONDITION [<filter_index>] <and_
↪expression_0> <and_not_expression_0> <and_expression_1> <and_not_
↪expression_1> <and_expression_2> <and_expression_3>

# get
<module-index>/<port-index> PF_CONDITION [<filter_index>] ?
```

Description

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 or 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 `[match_term_index] = 2^match_term_index`

Value for length term `[length_term_index] = 2^(length_term_index+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.

At the top level, a condition is a bunch of things or-ed together.

`<filter-condition> = <or-expr>`

Two of the or-operands are *general*, two are 'simple'.

```
<or-expr> = <general-and-expr> or <general-and-expr> or
<simple-and-expr> or <simple-and-expr>
```

A 'general' and-expression can include negated terms.

```
<general-and-expr> = <term> and <term> and ... and not <term> and ...
and not <term>
```

A 'simple' and-expression can only have non-negated terms.

`<simple-and-expr> = <term> and <term> and ... and <term>`

`<term> = <match-term>`

<term> = <length-term>

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

Actions

set, get

Parameters

1. and_expression_0: *integer*, encoding a compound term that is a set of the match terms AND length terms.
2. and_not_expression_0: *integer*, encoding a compound term that is a set of the match NOT terms AND length NOT terms.
3. and_expression_1: *integer*, encoding a compound term that is a set of the match terms AND length terms.
4. and_not_expression_1: *integer*, encoding a compound term that is a set of the match NOT terms AND length NOT terms.
5. and_expression_2: *integer*, encoding a compound term that is a set of the match terms AND length terms.
6. and_expression_3: *integer*, encoding a compound term that is a set of the match terms AND length terms.

Example

```
# set
input:  0/1 PF_CONDITION [0] 1 1 1 1 1 1
output: <OK>

# get
input:  0/1 PF_CONDITION [0] ?
output: 0/1 PF_CONDITION [0] 1 1 1 1 1 1
```

PF_CONFIG

code: 218

```
# get
<module-index>/<port-index> PF_CONFIG [<filter_index>] ?
```

Description

Return all relevant values for filter

Actions

get

Parameters

Example

```
# get
input: 0/1 PF_CONFIG [0] ?
```

PF_CREATE

code: 212

```
# set
<module-index>/<port-index> PF_CREATE [<filter_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PF_CREATE [0]
output: <OK>
```

PF_DELETE

code: 213

```
# set
<module-index>/<port-index> PF_DELETE [<filter_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PF_DELETE [0]
output: <OK>
```

PF_ENABLE

code: 214

```
# set
<module-index>/<port-index> PF_ENABLE [<filter_index>] <on_off>

# get
<module-index>/<port-index> PF_ENABLE [<filter_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. on_off: *byte*, whether the filter is enabled
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 PF_ENABLE [0] OFF
output: <OK>

# get
input:  0/1 PF_ENABLE [0] ?
output: 0/1 PF_ENABLE [0] OFF
```

PF_INDICES

code: 211

```
# set
<module-index>/<port-index> PF_INDICES <filter_xindices>

# get
<module-index>/<port-index> PF_INDICES ?
```

Description

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.

Actions

set, get

Parameters

1. `filter_xindices`: *integer list*, the list of indices of filters to be created on a port.

Example

```
# set
input:  0/1 PF_INDICES 0 1
output: <OK>

# get
input:  0/1 PF_INDICES ?
output: 0/1 PF_INDICES 0 1
```

PF_STRING

code: 217

```
# set
<module-index>/<port-index> PF_STRING [<filter_index>] <string_name>

# get
<module-index>/<port-index> PF_STRING [<filter_index>] ?
```

Description

The string representation of a filter.

Actions

set, get

Parameters

1. `string_name`: *string*, the string representation of the filter

Example

```
# set
input:  0/1 PF_STRING [0] "a string"
output: <OK>

# get
input:  0/1 PF_STRING [0] ?
output: 0/1 PF_STRING [0] "a string"
```

Length Term

Port length term commands that deal with configuration of the length term on the received traffic of a port.

The length term command names all have the form `PL_<xxx>`, and require both a module index id and a port index id, as well as a sub-index identifying a particular length term.

The length terms provide basic true/false indications for each packet received on the port.

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.

PL_CREATE

code: 208

```
# set
<module-index>/<port-index> PL_CREATE [<length_term_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PL_CREATE [0]
output: <OK>
```

PL_DELETE

code: 209

```
# set
<module-index>/<port-index> PL_DELETE [<length_term_index>]
```

Description

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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PL_DELETE [0]
output: <OK>
```

PL_INDICES

code: 207

```
# set
<module-index>/<port-index> PL_INDICES <length_term_xindices>

# get
<module-index>/<port-index> PL_INDICES ?
```

Description

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.

Actions

set, get

Parameters

1. `length_term_xindices`: *integer list*, the list of indices of length terms to be created on a port.

Example

```
# set
input:  0/1 PL_INDICES 0 1
output: <OK>

# get
input:  0/1 PL_INDICES ?
output: 0/1 PL_INDICES 0 1
```

PL_LENGTH

`code:  210`

```
# set
<module-index>/<port-index> PL_LENGTH [<length_term_index>] <length_
↪check_type> <size>

# get
<module-index>/<port-index> PL_LENGTH [<length_term_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `length_check_type`: *byte*, whether to test for shorter-than or longer-than
   - `AT_MOST = 0`
   - `AT_LEAST = 1`
2. `size`: *integer*, the value to compare the packet length against

Example

```
# set
input:  0/1 PL_LENGTH [0] AT_MOST 1
output: <OK>

# get
input:  0/1 PL_LENGTH [0] ?
output: 0/1 PL_LENGTH [0] AT_MOST 1
```

Match Term

Port match term commands that deal with configuration of the length term on the received traffic of a port.

The match term command names all have the form `PM_<xxx>`, and require both a module index id and a port index id, as well as a sub-index identifying a particular match term.

The match terms provide basic true/false indications for each packet received on the port.

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_CONFIG

`code:  206`

```
# get
<module-index>/<port-index> PM_CONFIG [<match_term_index>] ?
```

Description

Return all values for match term

Actions

get

Parameters

Example

```
# get
input:  0/1 PM_CONFIG [0] ?
```

PM_CREATE

code: 201

```
# set
<module-index>/<port-index> PM_CREATE [<match_term_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PM_CREATE [0]
output: <OK>
```

PM_DELETE

code: 202

```
# set
<module-index>/<port-index> PM_DELETE [<match_term_index>]
```

Description

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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PM_DELETE [0]
output: <OK>
```

PM_INDICES

code: 200

```
# set
<module-index>/<port-index> PM_INDICES <match_term_xindices>

# get
<module-index>/<port-index> PM_INDICES ?
```

Description

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.

Actions

set, get

Parameters

1. `match_term_xindices`: *integer list*, the sub-index of a match term definition for the port

Example

```
# set
input:  0/1 PM_INDICES 0 1
output: <OK>

# get
input:  0/1 PM_INDICES ?
output: 0/1 PM_INDICES 0 1
```

PM_MATCH

`code:  205`

```
# set
<module-index>/<port-index> PM_MATCH [<match_term_index>] <mask>
↪<value>

# get
<module-index>/<port-index> PM_MATCH [<match_term_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. `mask`: *hex8*, which bits are significant in the match operation
2. `value`: *hex8*, the value that must be found for the match term to be true

Example

```
# set
input:  0/1 PM_MATCH [0] 0xFF00000000000000 0x0000000000000000
output: <OK>

# get
input:  0/1 PM_MATCH [0] ?
output: 0/1 PM_MATCH [0] 0xFF00000000000000 0x0000000000000000
```

PM_POSITION

`code:  204`

```
# set
<module-index>/<port-index> PM_POSITION [<match_term_index>] <byte_
↪offset>

# get
<module-index>/<port-index> PM_POSITION [<match_term_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `byte_offset`: *integer*, offset from the start of the packet bytes

Example

```
# set
input:  0/1 PM_POSITION [0] 1
output: <OK>

# get
input:  0/1 PM_POSITION [0] ?
output: 0/1 PM_POSITION [0] 1
```

PM_PROTOCOL

code: 203

```
# set
<module-index>/<port-index> PM_PROTOCOL [<match_term_index>] <segments>

# get
<module-index>/<port-index> PM_PROTOCOL [<match_term_index>] ?
```

Description

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 of the match position is specified with `PM_POSITION`.

Actions

set, get

Parameters

1. `segments`: *byte list*, a number specifying a built-in protocol segment: Uses the same coded values as the *PS_HEADERPROTOCOL* parameter

- `ETHERNET = 1`
- `VLAN = 2`
- `ARP = 3`
- `IP = 4`
- `IPV6 = 5`
- `UDP = 6`
- `TCP = 7`
- `LLC = 8`
- `SNAP = 9`
- `GTP = 10`
- `ICMP = 11`
- `RTP = 12`
- `RTCP = 13`
- `STP = 14`
- `SCTP = 15`
- `MACCTRL = 16`
- `MPLS = 17`
- `PBBTAG = 18`
- `FCOE = 19`
- `FC = 20`
- `FCOETAIL = 21`
- `IGMPV3L0 = 22`
- `IGMPV3L1 = 23`
- `UDPCHECK = 24`
- `IGMPV2 = 25`
- `MPLS_TP_OAM = 26`

- GRE_NOCHECK = 27
- GRE_CHECK = 28
- TCPCHECK = 29
- GTPV1L0 = 30
- GTPV1L1 = 31
- GTPV2L0 = 32
- GTPV2L1 = 33
- IGMPV1 = 34
- PWETHCTRL = 35
- VXLAN = 36
- ETHERNET_8023 = 37
- NVGRE = 38
- DHCPV4 = 39
- GENEVE = 40
- XENA_TPLD = 41
- XENA_TPLD_PI = 42
- XENA_MICROTPLD = 43
- ETHERNET_FCS = 44
- MACCTRLPFC = 45
- ECPRI = 46
- ROE = 47
- ETHERTYPE = 48
- -n (n bytes custom segment)

Example

```
# set
input:  0/1 PM_PROTOCOL [0] ETHERNET VLAN IP -4
output: <OK>

# get
input:  0/1 PM_PROTOCOL [0] ?
output: 0/1 PM_PROTOCOL [0] ETHERNET VLAN IP -4
```

Capture

Port capture commands that 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 command. 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 command names all have the form PC_<xxx> and require both a module index id and a port index id. The per-packet parameters also use a sub-index identifying a particular packet in the capture buffer.

P_CAPTURE

code: 125

```
# set
<module-index>/<port-index> P_CAPTURE <on_off>

# get
<module-index>/<port-index> P_CAPTURE ?
```

Description

Whether a port is capturing packets. When on, the port retains the received packets and makes them available for inspection. The capture criteria are configured using the PC_xxx parameters. While capture is on the capture parameters cannot be changed.

Actions

set, get

Parameters

1. on_off: *byte*, whether the port is capturing packets.

- STOP = 0
- START = 1

Example

```
# set
input:  0/1 P_CAPTURE STOP
output: <OK>

# get
input:  0/1 P_CAPTURE ?
output: 0/1 P_CAPTURE STOP
```

PC_EXTRA

code: 225

```
# get
<module-index>/<port-index> PC_EXTRA [<capture_packet_index>] ?
```

Description

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.

Actions

get

Parameters

1. `time_captured`: *long integer*, time of capture, the number of nanoseconds since the packet was transmitted, the number of byte-times since previous packet, and the real length of the packet on the wire.
2. `latency`: *long integer*, time of capture, the number of nanoseconds since the packet was transmitted, the number of byte-times since previous packet, and the real length of the packet on the wire.
3. `byte_time_count`: *long integer*, time of capture, the number of nanoseconds since the packet was transmitted, the number of byte-times since previous packet, and the real length of the packet on the wire.
4. `length`: *integer*, time of capture, the number of nanoseconds since the packet was transmitted, the number of byte-times since previous packet, and the real length of the packet on the wire.

Example

```
# get
input:  0/1 PC_EXTRA [0] ?
output: 0/1 PC_EXTRA [0] 123456789123 123456789123 123456789123 1
```

PC_KEEP

code: 222

```
# set
<module-index>/<port-index> PC_KEEP <type> <index> <byte_count>

# get
<module-index>/<port-index> PC_KEEP ?
```

Description

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.

Actions

set, get

Parameters

1. `type`: *byte*, which general types of packets to keep
   - `ALL = 0`
   - `FCSERR = 1`
   - `NOTPLD = 2`
   - `TPLD = 3`
   - `FILTER = 4`
   - `PLDERR = 5`
2. `index`: *integer*, test payload id or filter index for which packets to keep
3. `byte_count`: *integer*, how many bytes to keep in the buffer for of each packet. The value -1 means no limit on packet size.

Example

```
# set
input:  0/1 PC_KEEP ALL 1 1
output: <OK>

# get
input:  0/1 PC_KEEP ?
output: 0/1 PC_KEEP ALL 1 1
```

PC_PACKET

code: 226

```
# get
<module-index>/<port-index> PC_PACKET [<capture_packet_index>] ?
```

Description

Obtains the raw bytes of a captured packet for a port. The packet data may be truncated if the PC_KEEP command specified a limit on the number of bytes kept.

Actions

get

Parameters

1. `hex_data`: *hex list*, the raw bytes of a captured packet

Example

```
# get
input:  0/1 PC_PACKET [0] ?
output: 0/1 PC_PACKET [0] 0x57
```

PC_STATS

code: 224

```
# get
<module-index>/<port-index> PC_STATS ?
```

Description

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

Actions

get

Parameters

1. `status`: *long integer*, status of the capture, 1 if capture has been stopped because of overflow, 0 if still running
2. `packets`: *long integer*, number of packets in the buffer.
3. `start_time`: *long integer*, start time in **nanoseconds** since 2010-01-01.

Example

```
# get
input:  0/1 PC_STATS ?
output: 0/1 PC_STATS 1 100 123456789123
```

PC_TRIGGER

code: 221

```
# set
<module-index>/<port-index> PC_TRIGGER <start_criteria> <start_
↪criteria_filter> <stop_criteria> <stop_criteria_filter>

# get
<module-index>/<port-index> PC_TRIGGER ?
```

Description

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).

Actions

set, get

Parameters

1. `start_criteria`: *byte*, the criteria for starting the actual packet capture
   - `ON = 0`
   - `FCSERR = 1`
   - `FILTER = 2`
   - `PLDERR = 3`
2. `start_criteria_filter`: *integer*, the index of a particular filter for the start criteria
3. `stop_criteria`: *byte*, the criteria for stopping the actual packet capture
   - `FULL = 0`
   - `FCSERR = 1`
   - `FILTER = 2`
   - `PLDERR = 3`
   - `USERSTOP = 4`
4. `stop_criteria_filter`: *integer*, the index of a particular filter for the stop criteria

Example

```
# set
input:  0/1 PC_TRIGGER ON 1 FULL 1
output: <OK>

# get
```

(continues on next page)

(continued from previous page)

```
input:  0/1 PC_TRIGGER ?
output: 0/1 PC_TRIGGER ON 1 FULL 1
```

Error Handling

P_ERRORS

code: 335

```
# get
<module-index>/<port-index> P_ERRORS ?
```

Description

Obtains the total number of errors detected across all streams on the port, including lost packets, misorder events, and payload errors.

Note: FCS errors are included, which will typically lead to double-counting of lost packets.

Actions

get

Parameters

1. `error_count`: *long integer*, the total number of errors detected across all streams on the port, including lost packets, misorder events, and payload errors

Example

```
# get
input:  0/1 P_ERRORS ?
output: 0/1 P_ERRORS 123456789123
```

P_TRAFFICERR

code: 198

```
# get
<module-index>/<port-index> P_TRAFFICERR ?
```

Description

Obtain the traffic error which has occurred in the last *_TRAFFIC or C_TRAFFICSYNC command.

Actions

get

Parameters

1. error: *byte*, traffic error which has occurred in the last *_TRAFFIC or C_TRAFFICSYNC command

- NOT_PREPARED = 0
- RATE_LENGTH_ERROR = 1
- PREPARED_OK = 2

Example

```
# get
input:  0/1 P_TRAFFICERR ?
output: 0/1 P_TRAFFICERR NOT_PREPARED
```

Stream

This module contains the **stream commands** deal with configuration of the traffic streams transmitted from a test port. The stream command names all have the form PS_<xxx> and require both a module index id and a port index id, as well as a sub-index identifying a particular stream.

General Information

Enabling Traffic

Whether the port is actually transmitting packets is controlled both by the `P_TRAFFIC` command for the parent port and by the `PS_ENABLE` command for the stream.

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

Stream Test Payload Data (TPLD)

Each Xena test packet contains a special proprietary data area called the *Test Payload Data (TPLD)*, which contains various information about the packet and is identified by a *Test Payload ID (TID)*. The *TPLD* is located just before the Ethernet FCS and consists of the following sections:

Table 3.1: Default TPLD (20 or 22 bytes)

Field	Length	Explanation
Checksum (optional)	2 bytes	See the *note*.
Sequence Number	3 bytes	Packet sequence number used for loss and misordering detection.
Timestamp	4 bytes	Timestamp value used for latency measurements.
Test Payload ID (TID)	2 bytes	Test payload identifier used to identify the sending stream.
Payload Integrity Offset	1 bit	Offset in packet from where to calculate payload integrity.
First Packet Flag	1 bit	Set if this is the first packet after traffic is started.
Checksum Enabled	1 bit	Set if payload integrity checksum is used.
<reserved>	7 bits	
Payload Integrity Offset (MSB)	3 bits	Offset in packet from where to calculate payload integrity, MSB (bits 10:9:8)
Timestamp Decimals	4 bits	Additional decimals for the timestamp.
Checksum	8 bytes	TPLD integrity checksum.
Total TPLD Size	**20 or 22 bytes**	

Note: If the `P_CHECKSUM offset` (Payload Checksum Offset) is enabled on the parent port, then an additional 2-byte checksum field is inserted in the TPLD, just before the Sequence Number. This increases the total size of the TPLD to 22 bytes.

Table 3.2: Micro-TPLD (6 bytes)

Field	Length	Explanation
First Packet Flag	1 bit	Packet sequence number used for loss and misordering detection.
<reserved>	1 bit	
Test Payload ID (TID)	10 bits	Test payload identifier used to identify the sending stream.
Timestamp	28 bits	Timestamp value used for latency measurements.
Checksum	8 bits	TPLD integrity checksum (CRC-8)
Total Micro-TPLD Size	**6 bytes**	

The selection between the default TPLD and the micro-TPLD is done on the parent port. It is thus not possible to use different TPLD types for streams on the same port.

Disabling TPLD The TPLD function can also be completely disabled for any given stream by setting the *Test Payload ID (TID)* value for the stream to the value -1.

Minimum Packet Size Considerations

The stream will generally accept any configuration and attempt to transmit packets according to the configuration. In order for the various Xena stream features to work correctly certain aspects about the minimum packet size used must be observed.

The minimum packet size must obviously be large enough to accommodate the defined `protocol headers + the final Ethernet FCS field`.

If the *TPLD* function explained above is enabled then each packet must also be able to contain the *TPLD* area (20, 22 or 6 bytes depending on the configuration).

If the stream payload type is set to `Incrementing`, then an additional minimum payload area of 2 bytes is needed. Otherwise excessive payload errors will be reported. This is however not necessary if the `P_CHECKSUM offset` (Payload Checksum Offset) option is enabled on the parent port as this will override the payload integrity check implied by the `Incrementing` payload type.

Control

PS_CREATE

`code: 151`

```
# set
<module-index>/<port-index> PS_CREATE [<stream_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_CREATE [0]
output: <OK>
```

PS_DELETE

code: 152

```
# set
<module-index>/<port-index> PS_DELETE [<stream_index>]
```

Description

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

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_DELETE [0]
output: <OK>
```

PS_ENABLE

code: 153

```
# set
<module-index>/<port-index> PS_ENABLE [<stream_index>] <state>

# get
<module-index>/<port-index> PS_ENABLE [<stream_index>] ?
```

Description

This property determines if a stream contributes outgoing packets for a port. The value can be toggled between ON and SUPPRESS while traffic is enabled at the port level. Streams in the OFF state cannot be set to any other value while traffic is enabled. The sum of the rates of all enabled or suppressed streams must not exceed the effective port rate.

Actions

set, get

Parameters

1. `state`: *byte*, a stream state
   - `OFF = 0`
   - `ON = 1`
   - `SUPPRESS = 2`

Example

```
# set
input:  0/1 PS_ENABLE [0] OFF
output: <OK>

# get
input:  0/1 PS_ENABLE [0] ?
output: 0/1 PS_ENABLE [0] OFF
```

PS_INDICES

code: 150

```
# set
<module-index>/<port-index> PS_INDICES <stream_indices>

# get
<module-index>/<port-index> PS_INDICES ?
```

Description

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 command 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.

Actions

set, get

Parameters

1. stream_indices: *integer list*, the sub-indices of streams on the port

Example

```
# set
input:  0/1 PS_INDICES 0 1
output: <OK>

# get
input:  0/1 PS_INDICES ?
output: 0/1 PS_INDICES 0 1
```

Identification

PS_COMMENT

code: 155

```
# set
<module-index>/<port-index> PS_COMMENT [<stream_index>] <comment>

# get
<module-index>/<port-index> PS_COMMENT [<stream_index>] ?
```

Description

The description of a stream.

Actions

set, get

Parameters

1. comment: *string*, the description of the stream

Example

```
# set
input:  0/1 PS_COMMENT [0] "this is a comment"
output: <OK>

# get
input:  0/1 PS_COMMENT [0] ?
output: 0/1 PS_COMMENT [0] "this is a comment"
```

PS_TPLDID

code: 157

```
# set
<module-index>/<port-index> PS_TPLDID [<stream_index>] <test_payload_
↪identifier>

# get
<module-index>/<port-index> PS_TPLDID [<stream_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. `test_payload_identifier`: *integer*, the test payload identifier value. -1 = disable test payloads

Example

```
# set
input:  0/1 PS_TPLDID [0] 1
output: <OK>

# get
input:  0/1 PS_TPLDID [0] ?
output: 0/1 PS_TPLDID [0] 1
```

TX Profile

PS_BURST

code: 174

```
# set
<module-index>/<port-index> PS_BURST [<stream_index>] <size> <density>

# get
<module-index>/<port-index> PS_BURST [<stream_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. `size`: *integer*, the number of packets lumped together in a burst
2. `density`: *integer*, the percentage of the available spacing that is inserted between bursts

Example

```
# set
input:  0/1 PS_BURST [0] 10 100
output: <OK>

# get
input:  0/1 PS_BURST [0] ?
output: 0/1 PS_BURST [0] 10 100
```

PS_BURSTGAP

code: 183

```
# set
<module-index>/<port-index> PS_BURSTGAP [<stream_index>] <inter_packet_
↪gap> <inter_burst_gap>

# get
<module-index>/<port-index> PS_BURSTGAP [<stream_index>] ?
```

Description

When the port is in in Burst TX mode, this command defines the gap between packets in a burst (inter-packet gap) and the gap after a burst defined in one stream stops until a burst defined in the next stream starts (inter-burst gap).

Actions

set, get

Parameters

1. *inter_packet_gap* `: *integer*, Burst Inter Packet Gap (in bytes).
2. `inter_burst_gap`: *integer*, Inter Burst Gap (in bytes).

Example

```
# set
input:  0/1 PS_BURSTGAP [0] 1 1
output: <OK>

# get
input:  0/1 PS_BURSTGAP [0] ?
output: 0/1 PS_BURSTGAP [0] 1 1
```

PS_PACKETLIMIT

code: 154

```
# set
<module-index>/<port-index> PS_PACKETLIMIT [<stream_index>] <packet_
↪count>

# get
<module-index>/<port-index> PS_PACKETLIMIT [<stream_index>] ?
```

Description

Based on different port transmission mode, the meaning of this API is different. When Port TX Mode is set to NORMAL, STRICT UNIFORM or BURST: The number of packets that will be transmitted when traffic is started on a port. A value of 0 or -1 makes the stream transmit continuously. When Port TX Mode is set to SEQUENTIAL: The number of sequential packets sent before switching to the next stream. The minimum value is 1. The port will transmit continuously until the user stops the traffic.

Actions

set, get

Parameters

1. `packet_count`: *integer*, the number of packets

Example

```
# set
input:  0/1 PS_PACKETLIMIT [0] 1
output: <OK>

# get
input:  0/1 PS_PACKETLIMIT [0] ?
output: 0/1 PS_PACKETLIMIT [0] 1
```

PS_RATEFRACTION

code: 169

```
# set
<module-index>/<port-index> PS_RATEFRACTION [<stream_index>] <stream_
↪rate_ppm>

# get
<module-index>/<port-index> PS_RATEFRACTION [<stream_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `stream_rate_ppm`: *integer*, stream rate expressed as a ppm value between 0 and 1,000,000.

Example

```
# set
input:  0/1 PS_RATEFRACTION [0] 1000000
output: <OK>

# get
input:  0/1 PS_RATEFRACTION [0] ?
output: 0/1 PS_RATEFRACTION [0] 1000000
```

PS_RATEL2BPS

code: 171

```
# set
<module-index>/<port-index> PS_RATEL2BPS [<stream_index>] <l2_bps>

# get
<module-index>/<port-index> PS_RATEL2BPS [<stream_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `l2_bps`: *long integer*, stream rate expressed in bits per second

Example

```
# set
input:  0/1 PS_RATEL2BPS [0] 1000000
output: <OK>

# get
input:  0/1 PS_RATEL2BPS [0] ?
output: 0/1 PS_RATEL2BPS [0] 1000000
```

PS_RATEPPS

code: 170

```
# set
<module-index>/<port-index> PS_RATEPPS [<stream_index>] <stream_rate_
↪pps>

# get
<module-index>/<port-index> PS_RATEPPS [<stream_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `stream_rate_pps`: *integer*, stream rate expressed in packets per second

Example

```
# set
input:  0/1 PS_RATEPPS [0] 1000
output: <OK>

# get
input:  0/1 PS_RATEPPS [0] ?
output: 0/1 PS_RATEPPS [0] 1000
```

Connectivity Check

PS_ARPREQUEST

code: 161

```
# get
<module-index>/<port-index> PS_ARPREQUEST [<stream_index>] ?
```

Description

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 command does not generate an immediate result, but waits until an ARP reply is received on the test port. If no reply is received within 500 milliseconds, it returns.

Actions

get

Parameters

1. `mac_address`: *hex6*, the MAC address of the peer port

Example

```
# get
input:  0/1 PS_ARPREQUEST [0] ?
output: 0/1 PS_ARPREQUEST [0] 0x010203040506
```

PS_IPV4GATEWAY

code: 181

```
# set
<module-index>/<port-index> PS_IPV4GATEWAY [<stream_index>] <gateway>

# get
<module-index>/<port-index> PS_IPV4GATEWAY [<stream_index>] ?
```

Description

An IPv4 gateway configuration specified for a stream.

Actions

set, get

Parameters

1. `gateway`: *address*, the IPv4 gateway address of the stream

Example

```
# set
input:  0/1 PS_IPV4GATEWAY [0] 192.168.1.1
output: <OK>

# get
input:  0/1 PS_IPV4GATEWAY [0] ?
output: 0/1 PS_IPV4GATEWAY [0] 192.168.1.1
```

PS_IPV6GATEWAY

`code:  182`

```
# set
<module-index>/<port-index> PS_IPV6GATEWAY [<stream_index>] <gateway>

# get
<module-index>/<port-index> PS_IPV6GATEWAY [<stream_index>] ?
```

Description

An IPv6 gateway configuration specified for a stream.

Actions

set, get

Parameters

1. `gateway`: *string*, the IPv6 gateway address of the stream

Example

```
# set
input:  0/1 PS_IPV6GATEWAY [0] 192.168.1.1
output: <OK>

# get
input:  0/1 PS_IPV6GATEWAY [0] ?
output: 0/1 PS_IPV6GATEWAY [0] 192.168.1.1
```

PS_PINGREQUEST

`code:  162`

```
# get
<module-index>/<port-index> PS_PINGREQUEST [<stream_index>] ?
```

Description

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 command does not generate an immediate result, but waits until a ping reply is received on the test port.

Actions

get

Parameters

1. `delay`: *integer*, the number of milliseconds for the ping reply to arrive.
2. `time_to_live`: *short integer*, the time-to-live value in the ping reply packet.

Example

```
# get
input:  0/1 PS_PINGREQUEST [0] ?
output: 0/1 PS_PINGREQUEST [0] 1 123
```

Error Injection

PS_INJECTFCSERR

`code:  185`

```
# set
<module-index>/<port-index> PS_INJECTFCSERR [<stream_index>]
```

Description

Force a frame checksum error in one of the packets currently being transmitted from a stream. This can aid in analyzing the error-detection functionality of the system under test. Traffic must be on for the port, and the stream must be enabled.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_INJECTFCSERR [0]
output: <OK>
```

PS_INJECTMISERR

code: 187

```
# set
<module-index>/<port-index> PS_INJECTMISERR [<stream_index>]
```

Description

Force a misorder error by swapping the test payload sequence numbers in two of the packets currently being transmitted from a stream. This can aid in analyzing 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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_INJECTMISERR [0]
output: <OK>
```

PS_INJECTPLDERR

code: 188

```
# set
<module-index>/<port-index> PS_INJECTPLDERR [<stream_index>]
```

Description

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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_INJECTPLDERR [0]
output: <OK>
```

PS_INJECTSEQERR

code: 186

```
# set
<module-index>/<port-index> PS_INJECTSEQERR [<stream_index>]
```

Description

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 analyzing 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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_INJECTSEQERR [0]
output: <OK>
```

PS_INJECTTPLDERR

code: 189

```
# set
<module-index>/<port-index> PS_INJECTTPLDERR [<stream_index>]
```

Description

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.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_INJECTTPLDERR [0]
output: <OK>
```

PS_INSERTFCS

code: 158

```
# set
<module-index>/<port-index> PS_INSERTFCS [<stream_index>] <on_off>

# get
<module-index>/<port-index> PS_INSERTFCS [<stream_index>] ?
```

Description

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

Actions

set, get

Parameters

1. on_off: *byte*, whether frame checksums are inserted

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 PS_INSERTFCS [0] OFF
output: <OK>

# get
```

(continues on next page)

(continued from previous page)

```
input:  0/1 PS_INSERTFCS [0] ?
output: 0/1 PS_INSERTFCS [0] OFF
```

Protocol Segment

PS_HEADERPROTOCOL

code: 176

```
# set
<module-index>/<port-index> PS_HEADERPROTOCOL [<stream_index>]
↪<segments>

# get
<module-index>/<port-index> PS_HEADERPROTOCOL [<stream_index>] ?
```

Description

This command will inform the Xena tester how to interpret the packet header byte sequence specified with PS_PACKETHEADER. This is mainly for information purposes, and the stream will transmit the packet header bytes even if no protocol segments are specified. The Xena tester however support calculation of certain field values in hardware, such as the IP, TCP and UDP length and checksum fields. This allow the use of hardware modifiers for these protocol segments. In order for this function to work the Xena tester needs to know the type of each segment that precedes the segment where the hardware calculation is to be performed.

Actions

set, get

Parameters

1. `segments`: *byte list*, a number specifying a built-in protocol segment

- `ETHERNET = 1`
- `VLAN = 2`
- `ARP = 3`
- `IP = 4`
- `IPV6 = 5`
- `UDP = 6`

- TCP = 7
- LLC = 8
- SNAP = 9
- GTP = 10
- ICMP = 11
- RTP = 12
- RTCP = 13
- STP = 14
- SCTP = 15
- MACCTRL = 16
- MPLS = 17
- PBBTAG = 18
- FCOE = 19
- FC = 20
- FCOETAIL = 21
- IGMPV3L0 = 22
- IGMPV3L1 = 23
- UDPCHECK = 24
- IGMPV2 = 25
- MPLS_TP_OAM = 26
- GRE_NOCHECK = 27
- GRE_CHECK = 28
- TCPCHECK = 29
- GTPV1L0 = 30
- GTPV1L1 = 31
- GTPV2L0 = 32
- GTPV2L1 = 33
- IGMPV1 = 34
- PWETHCTRL = 35
- VXLAN = 36
- ETHERNET_8023 = 37
- NVGRE = 38

- DHCPV4 = 39
- GENEVE = 40
- XENA_TPLD = 41
- XENA_TPLD_PI = 42
- XENA_MICROTPLD = 43
- ETHERNET_FCS = 44
- MACCTRLPFC = 45
- ECPRI = 46
- ROE = 47
- ETHERTYPE = 48
- -n (n bytes custom segment)

Example

```
# set
input:  0/1 PS_HEADERPROTOCOL [0] ETHERNET VLAN IP -4
output: <OK>

# get
input:  0/1 PS_HEADERPROTOCOL [0] ?
output: 0/1 PS_HEADERPROTOCOL [0] ETHERNET VLAN IP -4
```

PS_PACKETHEADER

code: 175

```
# set
<module-index>/<port-index> PS_PACKETHEADER [<stream_index>] <hex_data>

# get
<module-index>/<port-index> PS_PACKETHEADER [<stream_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. `hex_data`: *hex list*, the raw bytes comprising the packet header

Example

```
# set
input:  0/1 PS_PACKETHEADER [0]␣
↪0xAAAAAAAAAAAA04F4BC9DE7008100000A08004500002A000000007FFF3AD600000000000000000
output: <OK>

# get
input:  0/1 PS_PACKETHEADER [0] ?
output: 0/1 PS_PACKETHEADER [0]␣
↪0xAAAAAAAAAAAA04F4BC9DE7008100000A08004500002A000000007FFF3AD600000000000000000
```

Packet Content

PS_AUTOADJUST

code: 159

```
# set
<module-index>/<port-index> PS_AUTOADJUST [<stream_index>]
```

Description

Executing PS_AUTOADJUST will adjust the packet length distribution (PS_PACKETLENGTH) of the stream:

(1) Set the type of packet length distribution (PS_PACKETLENGTH <length_type>) to FIXED.

(2) Set the lower limit on the packet length (PS_PACKETLENGTH <min_val>) to exactly fit the specified protocol headers, *TPLD* and *FCS* (but never set to less than 64).

(3) Set the payload type of packets transmitted for the stream (PS_PAYLOAD <payload_type>) to PATTERN.

(4) If necessary, also set the maximum number of header content bytes (P_MAXHEADERLENGTH <p_maxheaderlength_label> <max_header_length>) that can be freely specified for each generated stream of the port to a higher value, if needed to accommodate the header size of the stream (implicitly given by the PS_PACKETHEADER command).

(5) If the needed maximum header length (P_MAXHEADERLENGTH <p_maxheaderlength_label> <max_header_length>) is not possible with the actual number of active streams for the port, the command will fail with <BADVALUE>.

Actions

set

Parameters

Example

```
# set
input:  0/1 PS_AUTOADJUST [0]
output: <OK>
```

PS_CDFCOUNT

code: 196

```
# set
<module-index>/<port-index> PS_CDFCOUNT [<stream_index>] <cdf_count>

# get
<module-index>/<port-index> PS_CDFCOUNT [<stream_index>] ?
```

Description

This command is part of the Custom Data Field (CDF) feature. It controls the number of custom data fields available for each stream. You can set a different number of fields for each stream. Changing the field count value to a larger value will leave all existing fields intact. Changing the field count value to a smaller value will remove all existing fields with an index larger than or equal to the new count. The feature requires that the *P_PAYLOADMODE* command on the parent port has been set to CDF. This enables the feature for all streams on this port.

Actions

set, get

Parameters

1. `cdf_count`: *integer*, the number of CDF data fields to allocate for the stream

Example

```
# set
input:  0/1 PS_CDFCOUNT [0] 1
output: <OK>

# get
input:  0/1 PS_CDFCOUNT [0] ?
output: 0/1 PS_CDFCOUNT [0] 1
```

PS_CDFDATA

`code:  197`

```
# set
<module-index>/<port-index> PS_CDFDATA [<stream_index>, <custom_data_
↪field_index>] <hex_data>

# get
<module-index>/<port-index> PS_CDFDATA [<stream_index>, <custom_data_
↪field_index>] ?
```

Description

This command is part of the Custom Data Field (CDF) feature. It controls the actual field data for a single field. It is possible to define fields with different data lengths for each stream. If the length of a data field exceeds (packet length - CDF offset) defined for the stream the field data will be truncated when transmitted. The feature requires that the *P_PAYLOADMODE* command on the parent port has been set to `CDF`. This enables the feature for all streams on this port.

Actions

set, get

Parameters

1. `hex_data`: *hex list*, a pattern of bytes to be used

Example

```
# set
input:  0/1 PS_CDFDATA [0, 0] 0x3333333333333333
output: <OK>

# get
input:  0/1 PS_CDFDATA [0, 0] ?
output: 0/1 PS_CDFDATA [0, 0] 0x3333333333333333
```

PS_CDFOFFSET

`code:  195`

```
# set
<module-index>/<port-index> PS_CDFOFFSET [<stream_index>] <offset>

# get
<module-index>/<port-index> PS_CDFOFFSET [<stream_index>] ?
```

Description

This command is part of the Custom Data Field (CDF) feature. The CDF offset for the stream is the location in the stream data packets where the various CDF data will be inserted. All fields for a given stream uses the same offset value. The default value is 0, which means that the CDF data will be inserted at the very start of the packet, thus overwriting the packet protocol headers. If you want the CDF data to start immediately after the end of the packet protocol headers you will have to set the CDF field offset manually. The feature requires that the *P_PAYLOADMODE* command on the parent port has been set to CDF. This enables the feature for all streams on this port.

Actions

set, get

Parameters

1. `offset`: *integer*, the location where the CDF data will be inserted

Example

```
# set
input:  0/1 PS_CDFOFFSET [0] 1
output: <OK>

# get
input:  0/1 PS_CDFOFFSET [0] ?
output: 0/1 PS_CDFOFFSET [0] 1
```

PS_EXTPAYLOAD

`code:  199`

```
# set
<module-index>/<port-index> PS_EXTPAYLOAD [<stream_index>] <hex_data>

# get
<module-index>/<port-index> PS_EXTPAYLOAD [<stream_index>] ?
```

Description

This command controls the extended payload feature. The PS_PAYLOAD command described above only allow the user to specify an 18-byte pattern (when PS_PAYLOAD is set to `PATTERN`). The `PS_EXTPAYLOAD` command allow the definition of a much larger (up to MTU) payload buffer for each stream. The extended payload will be inserted immediately after the end of the protocol segment area. The feature requires the *P_PAYLOADMODE* command on the parent port being set to `EXTPL`. This enables the feature for all streams on this port.

Actions

set, get

Parameters

1. `hex_data`: *hex list*, the extended payload in bytes of a stream

Example

```
# set
input:  0/1 PS_EXTPAYLOAD [0]␣
↪0x123AA123BB123CC123AA123BB123CC123AA123BB123CC
output: <OK>

# get
input:  0/1 PS_EXTPAYLOAD [0] ?
output: 0/1 PS_EXTPAYLOAD [0]␣
↪0x123AA123BB123CC123AA123BB123CC123AA123BB123CC
```

PS_OPTIONS

`code:  220`

```
# set
<module-index>/<port-index> PS_OPTIONS [<stream_index>] <options>

# get
<module-index>/<port-index> PS_OPTIONS [<stream_index>] ?
```

Description

Define the set of active "option flags" for the stream. The "set" form sets the flags listed in <options>, and clears the flags not listed. To clear all flags, simply omit <options> in the command.

Actions

set, get

Parameters

1. `options`: *byte list*

- `INCPLDFROM0 = 0` This flag affects the `INC8/DEC8/INC16/DEC16` payload types (refer to the `PS_PAYLOAD` command): With the flag set, the first payload byte/word after the header will be 0 (`INC8/INC16`) or -1 (`DEC8/DEC16`). With the flag unset, the default is used: The first payload byte/word of the payload will be equal to `<length of header>` (`INC8/INC16`), or `-<length of header> - 1` (`DEC8/DEC16`).

Example

```
# set
input:  0/1 PS_OPTIONS [0] 0
output: <OK>

# get
input:  0/1 PS_OPTIONS [0] ?
output: 0/1 PS_OPTIONS [0] 0
```

PS_PACKETLENGTH

code: 179

```
# set
<module-index>/<port-index> PS_PACKETLENGTH [<stream_index>] <length_
↪type> <min_val> <max_val>

# get
<module-index>/<port-index> PS_PACKETLENGTH [<stream_index>] ?
```

Description

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.

Actions

set, get

Parameters

1. `length_type`: *integer*, the type of distribution of packet length
   - `FIXED = 0`
   - `INCREMENTING = 1`
   - `BUTTERFLY = 2`
   - `RANDOM = 3`
   - `MIX = 4`
2. `min_val`: *integer*, lower limit on the packet length
3. `max_val`: *integer*, upper limit on the packet length

Example

```
# set
input:  0/1 PS_PACKETLENGTH [0] FIXED 1 1
output: <OK>

# get
input:  0/1 PS_PACKETLENGTH [0] ?
output: 0/1 PS_PACKETLENGTH [0] FIXED 1 1
```

PS_PAYLOAD

code: 180

```
# set
<module-index>/<port-index> PS_PAYLOAD [<stream_index>] <payload_type>
 ↪<hex_data>

# get
<module-index>/<port-index> PS_PAYLOAD [<stream_index>] ?
```

Description

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

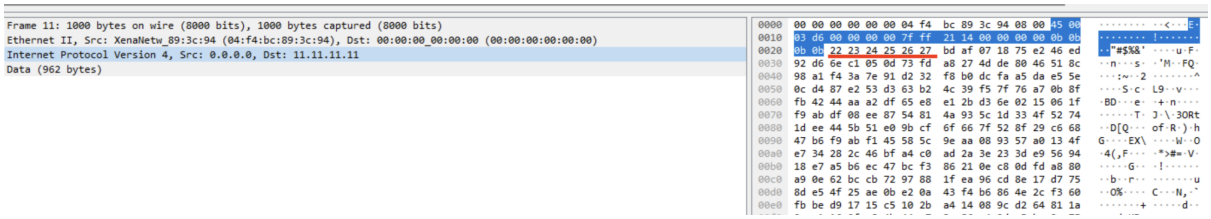
Actions

set, get

Parameters

1. `payload_type`: *byte*, the type of payload content
   - `PATTERN = 0`, a pattern is repeated up through the packet.
   - `INC8 = INCREMENTING = 1`, Incrementing Byte, 8-bit value, bytes are incremented up through the packet.
   - `PRBS = 2`, bytes are randomized from packet to packet.
   - `RANDOM = 3`, a randomly generated pattern.
   - `DEC8 = DECREMENTING = 4`, Decrementing Byte, 8-bit value, bytes are decremented up through the packet.
   - `INC16 = 5`, Incrementing Word, 16-bit value, bytes are incremented up through the packet.
   - `DEC16 = 6`, Decrementing Word, 16-bit value, bytes are decremented up through the packet.
2. `hex_data`: *hex list*, a pattern of bytes to be repeated. The maximum length of the pattern is 18 bytes. Only used if the type is set to *PATTERN*.

Important: The payload randomization feature has a limitation when utilizing either the PRBS or RANDOM modes. In these cases, the port initiates a process where it adds padding to the header portion in order to make it a multiple of 8 bytes. Subsequently, the remainder of the payload after this padding is randomized. This padding process employs a straightforward incrementing pattern, resulting in a set of fixed bytes following the header in each packet. An illustrative example of this limitation is presented below. Following the IP header, you will observe the initial 6 bytes of the payload, represented as `22 23 24 25 26 27` (incrementing), serving as the padding. Following the padding section, the payload undergoes randomization.



Example

```
# set
input:  0/1 PS_PAYLOAD [0] PATTERN 0x000102030405060708090A0B0C0D0E0F
output: <OK>

# get
input:  0/1 PS_PAYLOAD [0] ?
output: 0/1 PS_PAYLOAD [0] PATTERN 0x000102030405060708090A0B0C0D0E0F
```

Modifier

PS_MODIFIER

code: 178

```
# set
<module-index>/<port-index> PS_MODIFIER [<stream_index>, <modifier_
↪index>] <position> <mask> <action> <repetition>

# get
<module-index>/<port-index> PS_MODIFIER [<stream_index>, <modifier_
↪index>] ?
```

Description

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 command 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.

Actions

set, get

Parameters

1. `position`: *integer*, the byte position from the start of the packet
2. `mask`: *hex2*, the mask specifying which bits to affect
3. `action`: *byte*, which action to perform on the affected bits
   - `INC = 0`
   - `DEC = 1`
   - `RANDOM = 2`
4. `repetition`: *integer*, how many times to repeat on each packet

Example

```
# set
input:  0/1 PS_MODIFIER [0, 0] 14 0xFFFF INC 1
output: <OK>

# get
input:  0/1 PS_MODIFIER [0, 0] ?
output: 0/1 PS_MODIFIER [0, 0] 14 0xFFFF INC 1
```

PS_MODIFIERCOUNT

code: 177

```
# set
<module-index>/<port-index> PS_MODIFIERCOUNT [<stream_index>]
 ↪<modifier_count>

# get
<module-index>/<port-index> PS_MODIFIERCOUNT [<stream_index>] ?
```

Description

The number of standard 16-bit modifiers active on the packet header of a stream. Each modifier is specified using PS_MODIFIER.

Actions

set, get

Parameters

1. `modifier_count`: *integer*, the number of modifiers for the stream

Example

```
# set
input:  0/1 PS_MODIFIERCOUNT [0] 1
output: <OK>

# get
input:  0/1 PS_MODIFIERCOUNT [0] ?
output: 0/1 PS_MODIFIERCOUNT [0] 1
```

PS_MODIFIEREXT

code: 190

```
# set
<module-index>/<port-index> PS_MODIFIEREXT [<stream_index>, <modifier_
↪index>] <position> <mask> <action> <repetition>

# get
<module-index>/<port-index> PS_MODIFIEREXT [<stream_index>, <modifier_
↪index>] ?
```

Description

An extended packet modifier for a stream header. The headers of each packet transmitted for the stream will be varied according to the modifier specification. The modifier acts on 32 bits and takes up the space for two 16-bit modifiers to do this. This command 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.

Actions

set, get

Parameters

1. `position`: *integer*, the byte position from the start of the packet. Cannot be < 1!
2. `mask`: *hex3*, the mask specifying which bits to affect
3. `action`: *byte*, which action to perform on the affected bits
   - `INC = 0`
   - `DEC = 1`
   - `RANDOM = 2`
4. `repetition`: *integer*, how many times to repeat on each packet. Note: For now the only value supported is 1.

Example

```
# set
input:  0/1 PS_MODIFIEREXT [0, 0] 14 0xFFFFFF INC 1
output: <OK>

# get
input:  0/1 PS_MODIFIEREXT [0, 0] ?
output: 0/1 PS_MODIFIEREXT [0, 0] 14 0xFFFFFF INC 1
```

PS_MODIFIEREXTCOUNT

code: 191

```
# set
<module-index>/<port-index> PS_MODIFIEREXTCOUNT [<stream_index>] <ext_
↪modifier_count>

# get
<module-index>/<port-index> PS_MODIFIEREXTCOUNT [<stream_index>] ?
```

Description

The number of extended 24-bit modifiers active on the packet header of a stream. Each modifier is specified using PS_MODIFIEREXT.

Actions

set, get

Parameters

1. `ext_modifier_count`: *integer*, the number of extended 24-bit modifiers for the stream

Example

```
# set
input:  0/1 PS_MODIFIEREXTCOUNT [0] 1
output: <OK>

# get
input:  0/1 PS_MODIFIEREXTCOUNT [0] ?
output: 0/1 PS_MODIFIEREXTCOUNT [0] 1
```

PS_MODIFIEREXTRANGE

code: 167

```
# set
<module-index>/<port-index> PS_MODIFIEREXTRANGE [<stream_index>,
↪<modifier_index>] <min_val> <step> <max_val>

# get
<module-index>/<port-index> PS_MODIFIEREXTRANGE [<stream_index>,
↪<modifier_index>] ?
```

Description

Range specification for an extended 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: mix, step, and max, where max must be equal to min plus a multiple of step. Note that when "decrement" is specified in PS_MODIFIEREXT as the action, the value sequence will begin with the max value instead of the min value and decrement from there: {max, max-1, max-2, ..., min, max, max-1... }.

Actions

set, get

Parameters

1. `min_val`: *integer*, the minimum modifier value
2. `step`: *integer*, the increment between modifier values
3. `max_val`: *integer*, the maximum modifier value

Example

```
# set
input:  0/1 PS_MODIFIEREXTRANGE [0, 0] 1 1 10
output: <OK>

# get
input:  0/1 PS_MODIFIEREXTRANGE [0, 0] ?
output: 0/1 PS_MODIFIEREXTRANGE [0, 0] 1 1 10
```

PS_MODIFIERRANGE

`code:  168`

```
# set
<module-index>/<port-index> PS_MODIFIERRANGE [<stream_index>,
↪<modifier_index>] <min_val> <step> <max_val>

# get
<module-index>/<port-index> PS_MODIFIERRANGE [<stream_index>,
↪<modifier_index>] ?
```

Description

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 three values: mix, step, and max, where max must be equal to min plus a multiple of step. Note that when "decrement" is specified in PS_MODIFIER as the action, the value sequence will begin with the max value instead of the min value and decrement from there: {max, max-1, max-2, ..., min, max, max-1... }.

Actions

set, get

Parameters

1. `min_val`: *integer*, the minimum modifier value
2. `step`: *integer*, the increment between modifier values
3. `max_val`: *integer*, the maximum modifier value

Example

```
# set
input:  0/1 PS_MODIFIERRANGE [0, 0] 1 1 10
output: <OK>

# get
input:  0/1 PS_MODIFIERRANGE [0, 0] ?
output: 0/1 PS_MODIFIERRANGE [0, 0] 1 1 10
```

Config

PS_CONFIG

`code:  184`

```
# get
<module-index>/<port-index> PS_CONFIG [<stream_index>] ?
```

Description

Return all relevant values for stream

Actions

get

Parameters

Example

```
# get
input:  0/1 PS_CONFIG [0] ?
```

PFC

PS_PFCPRIORITY

code: 219

```
# set
<module-index>/<port-index> PS_PFCPRIORITY [<stream_index>] <cos>

# get
<module-index>/<port-index> PS_PFCPRIORITY [<stream_index>] ?
```

Description

Set and get the Priority Flow Control (PFC) Cos value of a stream.

Actions

set, get

Parameters

1. cos: *byte*, the PFC CoS value of the stream.
   - 0: the PFC CoS value = 0.
   - 1: the PFC CoS value = 1.
   - 2: the PFC CoS value = 2.
   - 3: the PFC CoS value = 3.
   - 4: the PFC CoS value = 4.

- 5: the PFC CoS value = 5.
- 6: the PFC CoS value = 6.
- 7: the PFC CoS value = 7.
- VLAN_PCP = 128: the PFC CoS value is automatically using the outer VLAN PCP value of the stream. If the VLAN field is missing, the stream won't have a PFC CoS.
- NO_PRIO = 129: no PFC priority is assigned to the stream.

Example

```
# set
input:  0/1 PS_PFCPRIORITY [0] VLAN_PCP
output: <OK>

# get
input:  0/1 PS_PFCPRIORITY [0] ?
output: 0/1 PS_PFCPRIORITY [0] VLAN_PCP
```

Statistics

TX Statistics

Port TX statistics commands that provide quantitative information about the transmitted packets on a port.

The command names all have the form PT_<xxx> and require both a module index id and a port index id. Those commands dealing with a specific transmitted stream also have a sub-index.

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_CLEAR

code: 233

```
# set
<module-index>/<port-index> PT_CLEAR
```

Description

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

Actions

set

Parameters

none

Example

```
# set
input:  0/1 PT_CLEAR
output: <OK>
```

PT_EXTRA

code: 235

```
# get
<module-index>/<port-index> PT_EXTRA ?
```

Description

Obtains additional statistics for packets transmitted on a port.

Actions

get

Parameters

1. `tx_arp_req_count`: *long integer*, number of ARP requests transmitted.
2. `tx_arp_res_count`: *long integer*, number of ARP responses transmitted.
3. `tx_ping_req_count`: *long integer*, number of PING requests transmitted.
4. `tx_ping_res_count`: *long integer*, number of PING responses transmitted.
5. `tx_fcs_inj_count`: *long integer*, number of FCS errors injected.
6. `tx_seq_inj_count`: *long integer*, number of sequence mismatch errors injected.
7. `tx_mis_inj_count`: *long integer*, number of packet misordering errors injected.
8. `tx_pld_inj_count`: *long integer*, number of payload errors injected.
9. `tx_tpld_inj_count`: *long integer*, number of payload integrity errors injected.
10. `tx_train_inj_count`: *long integer*, number of MAC learning (training) packets transmitted.
11. `tx_igmp_pac_count`: *long integer*, number of IGMP JOIN packets transmitted.

Example

```
# get
input:  0/1 PT_EXTRA ?
output: 0/1 PT_EXTRA 0 0 0 0 0 0 0 0 0 0 0
```

PT_NOTPLD

`code:  231`

```
# get
<module-index>/<port-index> PT_NOTPLD ?
```

Description

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

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared.

Example

```
# get
input:  0/1 PT_NOTPLD ?
output: 0/1 PT_NOTPLD 80000 8000 10000 8000
```

PT_NOTPLDEXT

`code:  237`

```
# get
<module-index>/<port-index> PT_NOTPLDEXT ?
```

Description

`PT_NOTPLDEXT` is an extension to `PT_NOTPLD` that also includes a calculation of bytes transmitted in the last second. `PT_NOTPLDEXT` returns list of *long integers*; this list may be expanded in future software releases.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second, same as in `PT_NOTPLD`.
2. `byte_count_last_sec`: *long integer*, number of bytes transmitted in the last second.
3. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second, same as in `PT_NOTPLD`.
4. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared, same as in `PT_NOTPLD`.
5. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared, same as in `PT_NOTPLD`.

Example

```
# get
input:  0/1 PT_NOTPLDEXT ?
output: 0/1 PT_NOTPLDEXT 80000 10000 8000 10000 8000
```

PT_STREAM

`code:  232`

```
# get
<module-index>/<port-index> PT_STREAM [<stream_index>] ?
```

Description

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

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared.

Example

```
# get
input:  0/1 PT_STREAM [0] ?
output: 0/1 PT_STREAM [0] 80000 8000 10000 8000
```

PT_STREAMEXT

`code: 238`

```
# get
<module-index>/<port-index> PT_STREAMEXT [<stream_index>] ?
```

Description

`PT_STREAMEXT` is an extension to `PT_STREAM` that also includes a calculation of bytes transmitted in the last second. `PT_STREAMEXT` returns list of *long integers*; this list may be expanded in future software releases.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second, same as in `PT_STREAM`.
2. `byte_count_last_sec`: *long integer*, number of bytes transmitted in the last second.
3. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second, same as in `PT_STREAM`.
4. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared, same as in `PT_STREAM`.
5. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared, same as in `PT_STREAM`.

Example

```
# get
input:  0/1 PT_STREAMEXT [0] ?
output: 0/1 PT_STREAMEXT [0] 80000 10000 8000 10000 8000
```

PT_TOTAL

`code:  230`

```
# get
<module-index>/<port-index> PT_TOTAL ?
```

Description

Obtains statistics concerning all the packets transmitted on a port.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared.

Example

```
# get
input:  0/1 PT_TOTAL ?
output: 0/1 PT_TOTAL 80000 8000 10000 8000
```

PT_TOTALEXT

`code:  236`

```
# get
<module-index>/<port-index> PT_TOTALEXT ?
```

Description

`PT_TOTALEXT` is an extension to `PT_TOTAL` that also includes a calculation of bytes transmitted in the last second. `PT_TOTALEXT` returns list of *long integers*; this list may be expanded in future software releases.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits transmitted in the last second, same as in `PT_TOTAL`.
2. `byte_count_last_sec`: *long integer*, number of bytes transmitted in the last second.
3. `packet_count_last_sec`: *long integer*, number of packets transmitted in the last second, same as in `PT_TOTAL`.
4. `byte_count_since_cleared`: *long integer*, number of bytes transmitted since statistics were cleared, same as in `PT_TOTAL`.
5. `packet_count_since_cleared`: *long integer*, number of packets transmitted since statistics were cleared, same as in `PT_TOTAL`.

Example

```
# get
input:  0/1 PT_TOTALEXT ?
output: 0/1 PT_TOTALEXT 80000 10000 8000 10000 8000
```

RX Statistics

Port RX statistics commands that provide quantitative information about the received packets on a port.

The command names all have the form `PR_<xxx>` and require both a module index id and a port index id. Those commands dealing with a specific received test payload id and a specific filter also have a sub-index id.

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.

PR_CALIBRATE

code: 249

```
# set
<module-index>/<port-index> PR_CALIBRATE
```

Description

Calibrate the latency calculation for packets received on a port. The lowest detected latency value (across all Test Payload IDs) will be set as the new base.

Actions

set

Parameters

none

Example

```
# set
input:  0/1 PR_CALIBRATE
output: <OK>
```

PR_CLEAR

code: 248

```
# set
<module-index>/<port-index> PR_CLEAR
```

Description

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

Actions

set

Parameters

none

Example

```
# set
input:  0/1 PR_CLEAR
output: <OK>
```

PR_EXTRA

code: 242

```
# get
<module-index>/<port-index> PR_EXTRA ?
```

Description

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

Actions

get

Parameters

1. `fcs_error_count`: *long integer*, number of packets with fcs error frames.
2. `pause_frame_count`: *long integer*, number of pause frames.
3. `rx_arp_request_count`: *long integer*, number of received ARP requests.
4. `rx_arp_reply_count`: *long integer*, number of received ARP responses.
5. `rx_ping_request_count`: *long integer*, number of received ping requests.
6. `rx_ping_reply_count`: *long integer*, number of received ping responses.
7. `gap_count`: *long integer*, number of gaps detected.
8. `gap_duration`: *long integer*, gap duration in microseconds.

Example

```
# get
input:  0/1 PR_EXTRA ?
output: 0/1 PR_EXTRA 0 0 4 4 5 6 0 0
```

PR_FILTER

`code:  247`

```
# get
<module-index>/<port-index> PR_FILTER [<filter_index>] ?
```

Description

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

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits received in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets received in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes received since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, number of packets received since statistics were cleared.

Example

```
# get
input:  0/1 PR_FILTER [0] ?
output: 0/1 PR_FILTER [0] 80000 8000 10000 8000
```

PR_FILTEREXT

`code:  260`

```
# get
<module-index>/<port-index> PR_FILTEREXT [<filter_index>] ?
```

Description

PR_FILTEREXT is an extension of `PR_FILTER` that also includes a calculation of bytes received in the last second. PR_FILTEREXT returns list of *long integers*. This list may be expanded in future software releases.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits received in the last second, same as in `PR_FILTER`.
2. `byte_count_last_sec`: *long integer*, number of bytes received in the last second.
3. `packet_count_last_sec`: *long integer*, number of packets received in the last second, same as in `PR_FILTER`.

4. `byte_count_since_cleared`: *long integer*, number of bytes received since statistics were cleared, same as in `PR_FILTER`.
5. `packet_count_since_cleared`: *long integer*, and number of packets received since statistics were cleared, same as in `PR_FILTER`.

Example

```
# get
input:  0/1 PR_FILTEREXT [0] ?
output: 0/1 PR_FILTEREXT [0] 80000 10000 8000 10000 8000
```

PR_NOTPLD

`code:  241`

```
# get
<module-index>/<port-index> PR_NOTPLD ?
```

Description

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

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits received in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets received in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes received since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, and number of packets received since statistics were cleared.

Example

```
# get
input:  0/1 PR_NOTPLD ?
output: 0/1 PR_NOTPLD 80000 8000 10000 8000
```

PR_NOTPLDEXT

code: 258

```
# get
<module-index>/<port-index> PR_NOTPLDEXT ?
```

Description

PR_NOTPLDEXT is an extension of PR_NOTPLD that also includes a calculation of bytes received in the last second. PR_NOTPLDEXT returns list of *long integers*. This list may be expanded in future software releases.

Actions

get

Parameters

1. bit_count_last_sec: *long integer*, number of bits received in the last second, same as in PR_NOTPLD.
2. byte_count_last_sec: *long integer*, number of bytes received in the last second.
3. packet_count_last_sec: *long integer*, number of packets received in the last second, same as in PR_NOTPLD.
4. byte_count_since_cleared: *long integer*, number of bytes received since statistics were cleared, same as in PR_NOTPLD.
5. packet_count_since_cleared: *long integer*, and number of packets received since statistics were cleared, same as in PR_NOTPLD.

Example

```
# get
input:  0/1 PR_NOTPLDEXT ?
output: 0/1 PR_NOTPLDEXT 80000 10000 8000 10000 8000
```

PR_PFCSTATS

code: 374

```
# get
<module-index>/<port-index> PR_PFCSTATS ?
```

Description

Obtains statistics of received Priority Flow Control (PFC) packets on a port.

Actions

get

Parameters

1. `packet_count`: *long integer*, the total number of Priority Flow Control (PFC) packets received since statistics were cleared
2. `quanta_prio_0`: *long integer*, the total number of valid PFC quanta received on the port for priority level 0 since statistics were cleared
3. `quanta_prio_1`: *long integer*, the total number of valid PFC quanta received on the port for priority level 1 since statistics were cleared
4. `quanta_prio_2`: *long integer*, the total number of valid PFC quanta received on the port for priority level 2 since statistics were cleared
5. `quanta_prio_3`: *long integer*, the total number of valid PFC quanta received on the port for priority level 3 since statistics were cleared
6. `quanta_prio_4`: *long integer*, the total number of valid PFC quanta received on the port for priority level 4 since statistics were cleared
7. `quanta_prio_5`: *long integer*, the total number of valid PFC quanta received on the port for priority level 5 since statistics were cleared
8. `quanta_prio_6`: *long integer*, the total number of valid PFC quanta received on the port for priority level 6 since statistics were cleared

9. `quanta_prio_7`: *long integer*, the total number of valid PFC quanta received on the port for priority level 7 since statistics were cleared

Example

```
# get
input:  0/1 PR_PFCSTATS ?
output: 0/1 PR_PFCSTATS 10 0 0 0 0 0 0 0 0
```

PR_TOTAL

`code:  240`

```
# get
<module-index>/<port-index> PR_TOTAL ?
```

Description

Obtains statistics concerning all the packets received on a port.

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits received in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets received in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes received since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, and number of packets received since statistics were cleared.

Example

```
# get
input:  0/1 PR_TOTAL ?
output: 0/1 PR_TOTAL 80000 8000 10000 8000
```

PR_TOTALEXT

code: 257

```
# get
<module-index>/<port-index> PR_TOTALEXT ?
```

Description

PR_TOTALEXT is an extension of PR_TOTAL that also includes a calculation of bytes received in the last second, as well as a number of port error counters. PR_TOTALEXT returns list of *long integers*. This list may be expanded in future software releases.

Actions

get

Parameters

1. bit_count_last_sec: *long integer*, number of bits received in the last second, same as in PR_TOTAL.
2. byte_count_last_sec: *long integer*, number of bytes received in the last second.
3. packet_count_last_sec: *long integer*, number of packets received in the last second, same as in PR_TOTAL.
4. byte_count_since_cleared: *long integer*, number of bytes received since statistics were cleared, same as in PR_TOTAL.
5. packet_count_since_cleared: *long integer*, and number of packets received since statistics were cleared, same as in PR_TOTAL.
6. fcs_error_count: *long integer*, number of packets received with fcs error frames, same as in PR_EXTRA.
7. oversize_count: *long integer*, number of oversize packets received since last clear; -1 if this counter is not supported by the tester.
8. undersize_count: *long integer*, number of undersize packets received since last clear; -1 if this counter is not supported by the tester.

9. jabber_count: *long integer*, number of jabber packets received since last clear; -1 if this counter is not supported by the tester.

Example

```
# get
input:  0/1 PR_TOTALEXT ?
output: 0/1 PR_TOTALEXT 80000 10000 8000 10000 8000 0 0 0 0
```

PR_TPLDERRORS

code: 245

```
# get
<module-index>/<port-index> PR_TPLDERRORS [<test_payload_index>] ?
```

Description

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.

Actions

get

Parameters

1. dummy: *long integer*, dummy value not in use.
2. non_incre_seq_event_count: *long integer*, number of non-incrementing-sequence-number events.
3. swapped_seq_misorder_event_count: *long integer*, number of swapped-sequence-number misorder events.
4. non_incre_payload_packet_count: *long integer*, number of packets with non-incrementing payload content.

Example

```
# get
input:  0/1 PR_TPLDERRORS [0] ?
output: 0/1 PR_TPLDERRORS [0] 0 0 0 0
```

PR_TPLDJITTER

code: 239

```
# get
<module-index>/<port-index> PR_TPLDJITTER [<test_payload_index>] ?
```

Description

Obtains statistics concerning the jitter experienced by the packets with a particular test payload id received on a port. The values are the difference in packet-to-packet latency, and the minimum will usually be zero.A special value of -1 is returned if jitter numbers are not applicable. They are only available for TID values 0..31.

Actions

get

Parameters

1. `min_val`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)
2. `avg_val`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)
3. `max_val`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)
4. `avg_last_sec`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)
5. `min_last_sec`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)
6. `max_last_sec`: *long integer*, minimum|average|maximum jitter (nanoseconds), average|average|maximum jitter over last 1-second period (nanoseconds)

Example

```
# get
input:  0/1 PR_TPLDJITTER [0] ?
output: 0/1 PR_TPLDJITTER [0] 8 9 10 9 8 10
```

PR_TPLDLATENCY

code: 246

```
# get
<module-index>/<port-index> PR_TPLDLATENCY [<test_payload_index>] ?
```

Description

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.

Actions

get

Parameters

1. min_val: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)
2. avg_val: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)
3. max_val: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)
4. avg_last_sec: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)
5. min_last_sec: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)
6. max_last_sec: *long integer*, minimum|average|maximum latency (nanoseconds), average|average|maximum latency over last 1-second period (nanoseconds)

Example

```
# get
input:  0/1 PR_TPLDLATENCY [0] ?
output: 0/1 PR_TPLDLATENCY [0] 8 9 10 9 8 10
```

PR_TPLDS

code: 243

```
# get
<module-index>/<port-index> PR_TPLDS ?
```

Description

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.

Actions

get

Parameters

1. `test_payload_identifiers`: *integer list*, the identifiers of the test payload

Example

```
# get
input:  0/1 PR_TPLDS ?
output: 0/1 PR_TPLDS 0 1
```

PR_TPLDTRAFFIC

code: 244

```
# get
<module-index>/<port-index> PR_TPLDTRAFFIC [<test_payload_index>] ?
```

Description

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

Actions

get

Parameters

1. `bit_count_last_sec`: *long integer*, number of bits received in the last second.
2. `packet_count_last_sec`: *long integer*, number of packets received in the last second.
3. `byte_count_since_cleared`: *long integer*, number of bytes received since statistics were cleared.
4. `packet_count_since_cleared`: *long integer*, number of packets received since statistics were cleared.

Example

```
# get
input:  0/1 PR_TPLDTRAFFIC [0] ?
output: 0/1 PR_TPLDTRAFFIC [0] 80000 8000 10000 8000
```

PR_TPLDTRAFFICEXT

code: 259

```
# get
<module-index>/<port-index> PR_TPLDTRAFFICEXT [<test_payload_index>] ?
```

Description

PR_TPLDTRAFFICEXT is an extension of PR_TPLDTRAFFIC that also includes a calculation of bytes received in the last second. PR_TPLDTRAFFICEXT returns list of *long integers*. This list may be expanded in future software releases.

Actions

get

Parameters

1. bit_count_last_sec: *long integer*, number of bits received in the last second, same as in PR_TPLDTRAFFIC.
2. byte_count_last_sec: *long integer*, number of bytes received in the last second.
3. packet_count_last_sec: *long integer*, number of packets received in the last second, same as in PR_TPLDTRAFFIC.
4. byte_count_since_cleared: *long integer*, number of bytes received since statistics were cleared, same as in PR_TPLDTRAFFIC.
5. packet_count_since_cleared: *long integer*, and number of packets received since statistics were cleared, same as in PR_TPLDTRAFFIC.

Example

```
# get
input:  0/1 PR_TPLDTRAFFICEXT [0] ?
output: 0/1 PR_TPLDTRAFFICEXT [0] 80000 10000 8000 10000 8000
```

Runt

P_RXPREAMBLE_INSERT

code: 394

Attention: Test module supported:

- Odin-10G-1S-6P[b]

```
# set
<module-index>/<port-index> P_RXPREAMBLE_INSERT <on_off>

# get
<module-index>/<port-index> P_RXPREAMBLE_INSERT ?
```

Description

Insert preambles to the incoming frames.

Actions

set, get

Parameters

1. `on_off`: *byte*, whether the port should insert preambles to the incoming frames
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 P_RXPREAMBLE_INSERT ON
output: <OK>

# get
input:  0/1 P_RXPREAMBLE_INSERT ?
output: 0/1 P_RXPREAMBLE_INSERT ON
```

P_RXRUNTLEN_ERRS

`code:  392`

Attention: Test module supported:

- Odin-10G-1S-6P[b]

```
# get
<module-index>/<port-index> P_RXRUNTLEN_ERRS ?
```

Description

Sticky clear on read: Have packets with wrong runt length been detected since last read?

Actions

get

Parameters

1. `status`: *byte*, whether packets with with wrong runt length been detected since last read
   - `NO = 0`
   - `YES = 1`

Example

```
# get
input:  0/1 P_RXRUNTLEN_ERRS ?
output: 0/1 P_RXRUNTLEN_ERRS NO
```

P_RXRUNTLENGTH

`code:  391`

> **Attention:** Test module supported:
>
> - Odin-10G-1S-6P[b]

```
# set
<module-index>/<port-index> P_RXRUNTLENGTH <runt_length>

# get
<module-index>/<port-index> P_RXRUNTLENGTH ?
```

Description

Enable RX runt length detection to flag if packets are seen with length other than <runt_length> bytes.

Actions

set, get

Parameters

1. runt_length: *integer*, RX runt length detection to flag if packets are seen with length not being <runt_length> bytes. Set to -1 to disabled.

Example

```
# set
input:  0/1 P_RXRUNTLENGTH 15
output: <OK>

# get
input:  0/1 P_RXRUNTLENGTH ?
output: 0/1 P_RXRUNTLENGTH 15
```

P_TXPREAMBLE_REMOVE

code: 393

Attention: Test module supported:

- Odin-10G-1S-6P[b]

```
# set
<module-index>/<port-index> P_TXPREAMBLE_REMOVE <on_off>

# get
<module-index>/<port-index> P_TXPREAMBLE_REMOVE ?
```

Description

Remove preamble from outgoing frames.

Actions

set, get

Parameters

1. `on_off`: *byte*, whether the preambles from outgoing frames are to be removed by the port
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 P_TXPREAMBLE_REMOVE ON
output: <OK>

# get
input:  0/1 P_TXPREAMBLE_REMOVE ?
output: 0/1 P_TXPREAMBLE_REMOVE ON
```

P_TXRUNTLENGTH

`code:  390`

Attention: Test module supported:

- Odin-10G-1S-6P[b]

```
# set
<module-index>/<port-index> P_TXRUNTLENGTH <runt_length>

# get
<module-index>/<port-index> P_TXRUNTLENGTH ?
```

Description

Enable TX runt feature to cut all packets to a number of bytes.

Actions

set, get

Parameters

1. `runt_length`: *integer*, enable TX runt feature to cut all packets to I bytes. Set to -1 to disable.

Example

```
# set
input:  0/1 P_TXRUNTLENGTH 15
output: <OK>

# get
input:  0/1 P_TXRUNTLENGTH ?
output: 0/1 P_TXRUNTLENGTH 15
```

A "runt" Ethernet frame is a term used in networking to describe an Ethernet frame that is too short to be valid according to Ethernet standards. Ethernet frames are supposed to adhere to a minimum length to ensure that the signal propagates properly across the network medium and can be reliably detected by receiving devices. The minimum frame length for Ethernet typically includes the data payload, frame header, and frame check sequence (FCS).

In a standard Ethernet frame, the minimum length is 64 bytes. If an Ethernet frame is shorter than this minimum length, it is considered a "runt" frame. Runt frames can be the result of various issues, such as collisions on the network, transmission errors, or improper frame encapsulation.

Runt frames are typically discarded by Ethernet network interfaces because they are too short to contain valid data. Discarding runt frames helps maintain network integrity and prevents unnecessary processing of invalid or corrupted data.

Port Runt commands allow you to "cut" packets into smaller "pieces".

As shown in Fig. 3.6, an Ethernet frame typically consists of the following components:

1. Preamble (7 bytes): The preamble is a sequence of 7 bytes (56 bits) used for synchronization and clock recovery. It helps the receiving Ethernet interface synchronize with the incoming data stream.
2. Start of Frame Delimiter (SFD) (1 byte): The SFD is a specific bit pattern (10101011) that marks the end of the preamble and the beginning of the frame's header.

3. Ethernet Header (Varies in length): The Ethernet header contains various fields, including the source and destination MAC addresses, EtherType (or Length), and sometimes additional information like VLAN tags or Quality of Service (QoS) information.
4. Data Payload (Varies in length): The data payload contains the actual data being transmitted in the Ethernet frame. Its length can vary depending on the type of Ethernet frame (e.g., Ethernet II, IEEE 802.3).
5. Frame Check Sequence (FCS) (4 bytes): The FCS is a checksum value calculated over the entire frame (header and data) to detect transmission errors. It helps ensure the integrity of the received frame.
6. Inter-Frame Gap (IFG) (12 bytes): The IFG is a gap or idle period between Ethernet frames, which ensures that there is enough time for network devices to process the incoming frame and prepare for the next one. It also helps prevent collisions in half-duplex Ethernet networks.
7. End of Frame (EOF): The EOF marks the end of the Ethernet frame and is used to signal the completion of the frame.

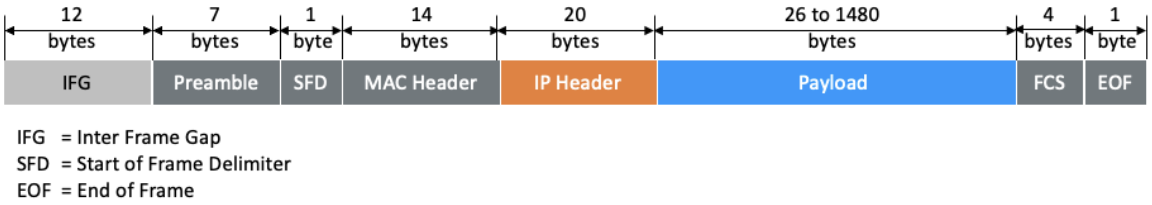


Fig. 3.6: Ethernet frame

The total length of an Ethernet frame can vary depending on factors like the Ethernet standard (e.g., Ethernet II, IEEE 802.3), whether it includes VLAN tags, and the size of the data payload. The minimum frame length for Ethernet is typically 64 bytes (as shown in Fig. 3.7), including all the components mentioned above, but longer frames are also allowed, with a maximum frame size specified by the Ethernet standard (e.g., 1518 bytes for standard Ethernet). Frames shorter than the minimum are considered "runt" frames and are often discarded as they may not provide sufficient time for network devices to operate correctly.

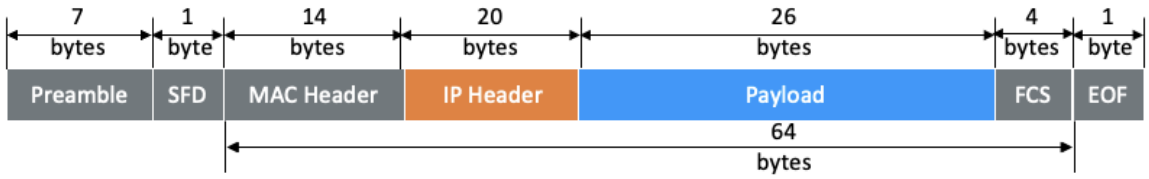


Fig. 3.7: 64-byte Ethernet frame

On the TX side, to "cut" an Ethernet frame into a runt frame, use *P_TXRUNTLENGTH*. The runt frame length starts from the MAC header as shown in Fig. 3.8. An EOF byte will be placed after the runt frame, and the rest becomes idle bytes as shown in Fig. 3.9. To remove the Preamble, use *P_TXPREAMBLE_REMOVE* with `off`, illustrated in Fig. 3.10.

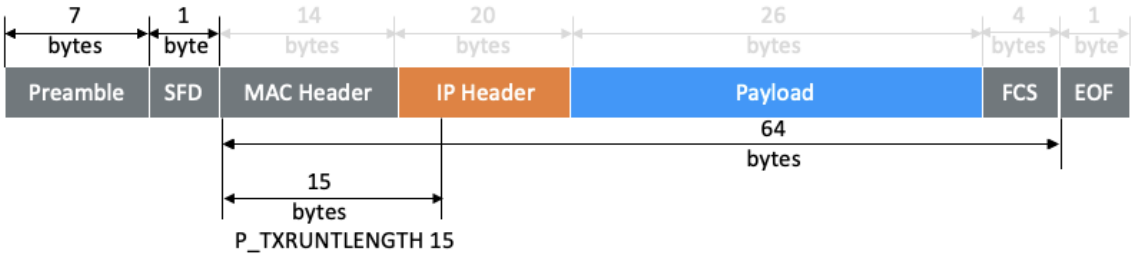


Fig. 3.8: Runt frame of 15 bytes on TX side

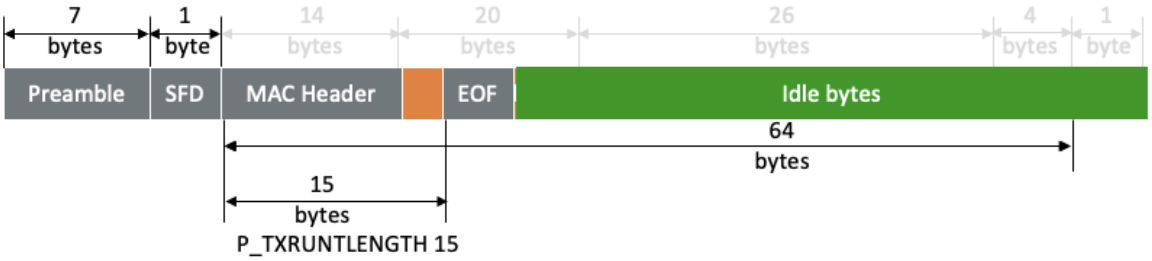


Fig. 3.9: An EOF byte will be placed after the runt frame, and the rest becomes idle bytes

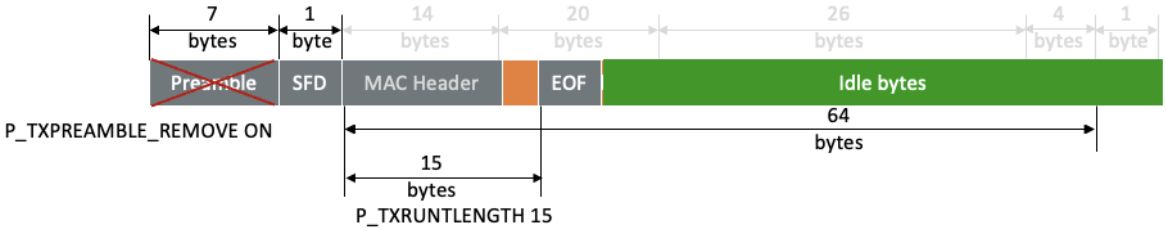


Fig. 3.10: Remove preamble on TX side

What is on the line will be a runt frame of the length you configured as illustrated in Fig. 3.11.

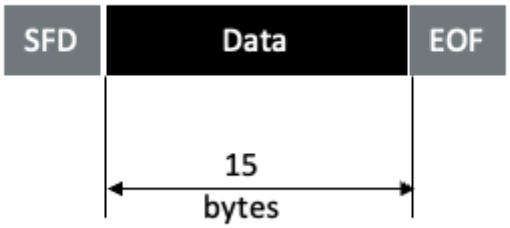


Fig. 3.11: What is on the line

On the RX side, as shown in Fig. 3.12, use *P_RXRUNTLENGTH* with the same value on the TX side so the port extends the packet size to 64 bytes by converting the "old" EOF byte to a data byte with value `FD` and also converting the following idle bytes to data bytes with value `07`. If other bytes than idle bytes are seen, they will be included in the 64-byte packet.

Use *P_TXPREAMBLE_REMOVE* with `on` to recover the preamble, which is required for upper layers to "see" the packet, as shown in Fig. 3.13.

Note: There is no valid FCS in the rebuild packet

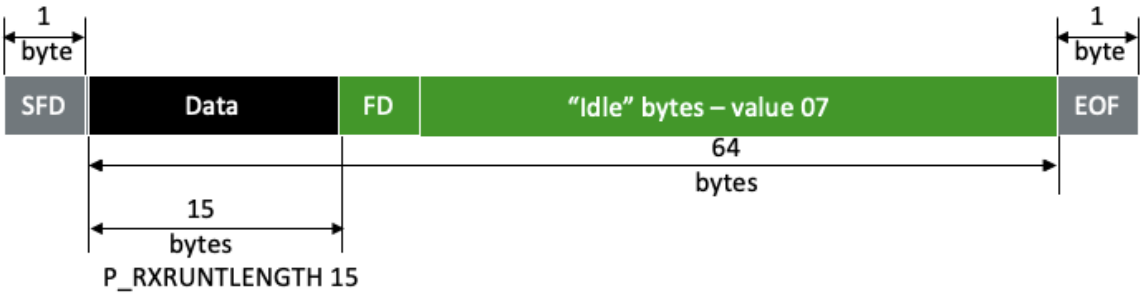


Fig. 3.12: Runt frame of 15 bytes on RX side

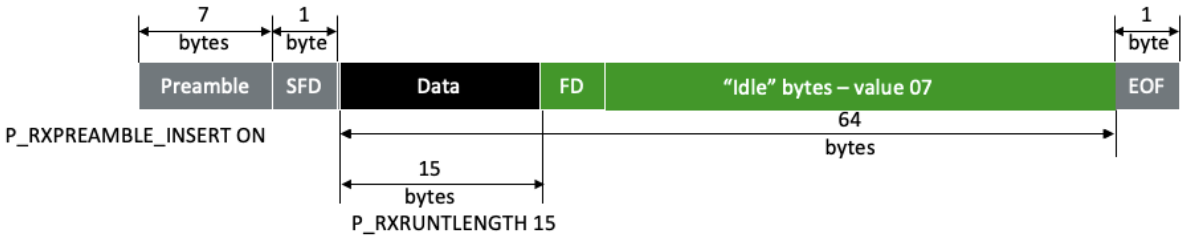


Fig. 3.13: Recover preamble on RX side

The capture in Fig. 3.14 shows what is on the RX side.

Important: Traffic rate must not exceed 10,500,000 packets per second at 10G line rate.

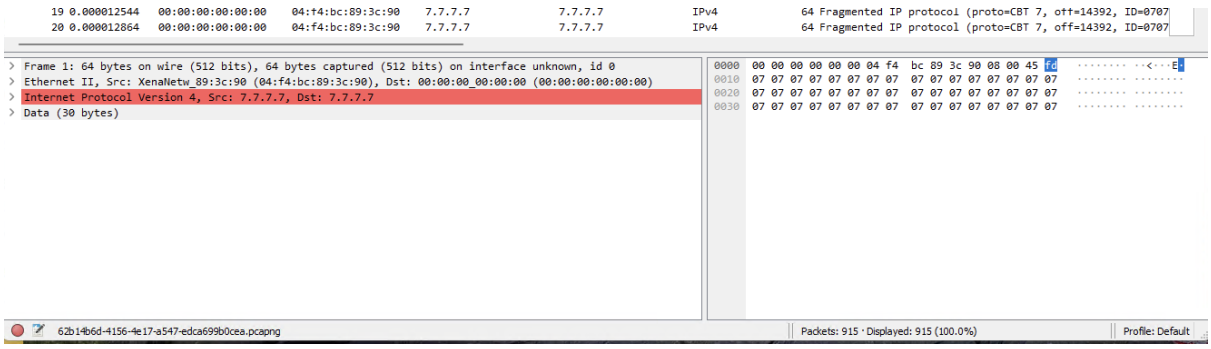


Fig. 3.14: Capture on RX side

At the receiver there must be minimum 80 bytes from start of one packet until start of next packet.

Packet sizes supported: 15-63 bytes.

Test module supported: Odin-10G-1S-6P[b]

Unavailable Time

P_UAT_FLR

code: 139

```
# set
<module-index>/<port-index> P_UAT_FLR <frame_loss_ratio>

# get
<module-index>/<port-index> P_UAT_FLR ?
```

Description

This command defines the threshold for the Frame Loss Ratio, where a second is declared as a Severely Errored Second (SES). In Valkyrie1564 UnAvailable Time (UAT) is declared after 10 consecutive SES has been detected

Actions

set, get

Parameters

1. `frame_loss_ratio`: *short integer*, Frame Loss Ratio specified as a number times 1/100, 0..100

Example

```
# set
input:  0/1 P_UAT_FLR 1
output: <OK>

# get
input:  0/1 P_UAT_FLR ?
output: 0/1 P_UAT_FLR 1
```

P_UAT_MODE

code: 138

```
# set
<module-index>/<port-index> P_UAT_MODE <mode> <delay>

# get
<module-index>/<port-index> P_UAT_MODE ?
```

Description

This command defines if a port is currently used by test suite Valkyrie1564, which means that UAT (UnAvailable Time) will be detected for the port.

Actions

set, get

Parameters

1. `mode`: *byte*, the state of the affected stream counters
   - `OFF = 0`
   - `ON = 1`
2. `delay`: *integer*, time in milliseconds to wait before detection of UAT is started. Default value: 500. This command is ignored when state is set to OFF

Example

```
# set
input:  0/1 P_UAT_MODE OFF 1
output: <OK>

# get
input:  0/1 P_UAT_MODE ?
output: 0/1 P_UAT_MODE OFF 1
```

PR_UAT_STATUS

code: 252

```
# get
<module-index>/<port-index> PR_UAT_STATUS ?
```

Description

This command will show the current UAT (UnAvailable Time) state, which is used in Valkyrie1564

Actions

get

Parameters

1. `status`: *byte*, the state of the affected stream counter

- `OFF = 0`
- `ON = 1`

Example

```
# get
input:  0/1 PR_UAT_STATUS ?
output: 0/1 PR_UAT_STATUS OFF
```

PR_UAT_TIME

code: 256

```
# get
<module-index>/<port-index> PR_UAT_TIME ?
```

Description

This command will show the current number of unavailable seconds, which is used in Valkyrie1564.

Actions

get

Parameters

1. `time`: *integer*, number of unavailable seconds

Example

```
# get
input:  0/1 PR_UAT_TIME ?
output: 0/1 PR_UAT_TIME 1
```

3.3.3 Layer-1

Layer-1 commands that provide configuration and status for the Gigabit Attachment Unit Interface (CAUI) physical coding sublayer used by 40G, 50G, 100G, 200G, 400G and 800G 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 id. The commands are indexed with a physical lane index that corresponds to a fixed numbering of the underlying fibers or wavelengths.

The lanes can also be put into *Pseudorandom Binary Sequence (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.

EEE

Energy Efficient Ethernet

P_LPENABLE

code: 340

```
# set
<module-index>/<port-index> P_LPENABLE <on_off>

# get
<module-index>/<port-index> P_LPENABLE ?
```

Description

Enables/disables Energy Efficient Ethernet (EEE) on the port.

Actions

set, get

Parameters

1. on_off: *byte*, whether Energy Efficient Ethernet (EEE) is enabled on the port
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_LPENABLE OFF
output: <OK>

# get
input:  0/1 P_LPENABLE ?
output: 0/1 P_LPENABLE OFF
```

P_LPPARTNERAUTONEG

code: 345

```
# get
<module-index>/<port-index> P_LPPARTNERAUTONEG ?
```

Description

Displays the EEE capabilities advertised during autonegotiation by the far side (link partner).

Actions

get

Parameters

1. `cap_100base_tx`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)
   - `NO = 0`
   - `YES = 1`
2. `cap_1000base_t`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)
   - `NO = 0`
   - `YES = 1`
3. `cap_10gbase_t`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)
   - `NO = 0`
   - `YES = 1`
4. `cap_100base_kx`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)
   - `NO = 0`
   - `YES = 1`
5. `cap_10gbase_kx4`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)
   - `NO = 0`
   - `YES = 1`

6. `cap_10gbase_kr`: *byte*, the the Energy Efficient Ethernet (EEE) capabilities advertised during autonegotiation by the far side (link partner)

- `NO = 0`
- `YES = 1`

Example

```
# get
input:  0/1 P_LPPARTNERAUTONEG ?
output: 0/1 P_LPPARTNERAUTONEG NO NO NO NO NO NO
```

P_LPRXPOWER

`code: 347`

```
# get
<module-index>/<port-index> P_LPRXPOWER ?
```

Description

Obtain the RX power recorded during training for the four channels.

Actions

get

Parameters

1. `channel_a`: *integer*, the the RX power recorded during training for the four channels
2. `channel_b`: *integer*, the RX power on link channel A
3. `channel_c`: *integer*, the RX power on link channel B
4. `channel_d`: *integer*, the RX power on link channel C

Example

```
# get
input:  0/1 P_LPRXPOWER ?
output: 0/1 P_LPRXPOWER 1 1 1 1
```

P_LPSNRMARGIN

code: 346

```
# get
<module-index>/<port-index> P_LPSNRMARGIN ?
```

Description

Displays the SNR margin on the four link channels (Channel A-D) as reported by the PHY. It is displayed in units of 0.1dB.

Actions

get

Parameters

1. *channel_a* `: *integer*, the SNR margin on the four link channels (Channel A-D) as reported by the PHY. It is displayed in units of 0.1dB
2. `channel_b`: *integer*, the SNR margin on the four link channels (Channel A-D) as reported by the PHY. It is displayed in units of 0.1dB
3. `channel_c`: *integer*, the SNR margin on the four link channels (Channel A-D) as reported by the PHY. It is displayed in units of 0.1dB
4. `channel_d`: *integer*, the SNR margin on the four link channels (Channel A-D) as reported by the PHY. It is displayed in units of 0.1dB

Example

```
# get
input:  0/1 P_LPSNRMARGIN ?
output: 0/1 P_LPSNRMARGIN 1 1 1 1
```

P_LPSTATUS

code: 343

```
# get
<module-index>/<port-index> P_LPSTATUS ?
```

Description

Displays the Energy Efficient Ethernet (EEE) status as reported by the PHY.

Actions

get

Parameters

1. `txh`: *byte*, the the Energy Efficient Ethernet (EEE) status
   - `TXH_NA = 0`
   - `TXH_X = 1`
2. `rxh`: *byte*, the the Energy Efficient Ethernet (EEE) status
   - `RXH_NA = 0`
   - `RXH_X = 1`
3. `txc`: *byte*, the the Energy Efficient Ethernet (EEE) status
   - `TXC_ACTIVE = 0`
   - `TXC_LPI = 1`
4. `rxc`: *byte*, the the Energy Efficient Ethernet (EEE) status
   - `RXC_ACTIVE = 0`
   - `RXC_LPI = 1`
5. `link_up`: *byte*, the the Energy Efficient Ethernet (EEE) status

- LINK_DOWN = 0
- LINK_UP = 1

Example

```
# get
input:  0/1 P_LPSTATUS ?
output: 0/1 P_LPSTATUS TXH_NA RXH_NA TXC_ACTIVE RXC_ACTIVE LINK_DOWN
```

P_LPSUPPORT

code: 351

```
# get
<module-index>/<port-index> P_LPSUPPORT ?
```

Description

Read EEE capabilities of the port (variable size, one for each supported speed, returns 0s if no EEE).

Actions

get

Parameters

1. `eee_capabilities`: *integer list*, the EEE capabilities of the port (variable size, one for each supported speed, returns 0s if no EEE).

Example

```
# get
input:  0/1 P_LPSUPPORT ?
output: 0/1 P_LPSUPPORT 0 1
```

P_LPTXMODE

code: 341

```
# set
<module-index>/<port-index> P_LPTXMODE <on_off>

# get
<module-index>/<port-index> P_LPTXMODE ?
```

Description

Enables/disables the transmission of Low Power Idles (LPIs) on the port. When enabled, the transmit side of the port will automatically enter low-power mode (and leave) low-power mode in periods of low or no traffic. LPIs will only be transmitted if the Link Partner (receiving port) has advertised EEE capability for the selected port speed during EEE auto-negotiation.

Actions

set, get

Parameters

1. on_off: *byte*, whether the transmission of Low Power Idles (LPIs) is enabled on the port
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_LPTXMODE OFF
output: <OK>

# get
input:  0/1 P_LPTXMODE ?
output: 0/1 P_LPTXMODE OFF
```

RS

Reconciliation Sublayer

P_FAULTSIGNALING

code: 348

```
# set
<module-index>/<port-index> P_FAULTSIGNALING <fault_signaling>

# get
<module-index>/<port-index> P_FAULTSIGNALING ?
```

Description

Sets the remote/local fault signaling behavior of the port (performed by the Reconciliation Sublayer). By default, the port acts according to the standard, i.e. when receiving a bad signal, it transmits "Remote Fault indications"on the output and when receiving a "Remote Fault indication"from the far-side it will transmit IDLE sequences.

Actions

set, get

Parameters

1. `fault_signaling`: *byte*, remote/local fault signaling behavior of the port

- `NORMAL = 0`
- `FORCE_LOCAL = 1`
- `FORCE_REMOTE = 2`
- `DISABLED = 3`

Example

```
# set
input:  0/1 P_FAULTSIGNALING NORMAL
output: <OK>

# get
input:  0/1 P_FAULTSIGNALING ?
output: 0/1 P_FAULTSIGNALING NORMAL
```

P_FAULTSTATUS

code: 349

```
# get
<module-index>/<port-index> P_FAULTSTATUS ?
```

Description

Shows if a local or remote fault is currently being detected by the Reconciliation Sub-layer of the port.

Note: Currently only available on M1CFP100, M2CFP40, M2QSFP+ and M1CFP4QSFP28CXP.

Actions

get

Parameters

1. `local_fault_status`: *byte*, whether a local or remote fault is currently being detected.
   - `OK = 0`
   - `LOCAL_FAULT = 1`
2. `remote_fault_status`: *byte*, specifying the local fault
   - `OK = 0`
   - `REMOTE_FAULT = 1`

Example

```
# get
input:  0/1 P_FAULTSTATUS ?
output: 0/1 P_FAULTSTATUS OK OK
```

PCS/FEC

Physical Coding Sublayer & Forward Error Correction

PL1_PCS_VARIANT

code: 434

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_PCS_VARIANT <variant>

# get
<module-index>/<port-index> PL1_PCS_VARIANT ?
```

Description

PCS variant configuration.

Actions

set, get

Parameters

1. `variant`: *byte*, PCS variant

- `IEEE = 1`
- `ETC = 2`

Example

```
# set
input:  0/1 PL1_PCS_VARIANT IEEE
output: <OK>

# get
input:  0/1 PL1_PCS_VARIANT ?
output: 0/1 PL1_PCS_VARIANT IEEE
```

PL1_CWE_BIT_ERR_MASK

`code:  437`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_BIT_ERR_MASK <mode> <bitmask>

# get
<module-index>/<port-index> PL1_CWE_BIT_ERR_MASK ?
```

Description

Configure the bit error mask for the errored symbols.

Actions

set, get

Parameters

1. `mode`: *integer*, bit error mask mode.

- `STATIC = 1`. The bit error pattern stay the same for all errored symbols.
- `ROTATE_HIGH = 2`. The bit error pattern shifts one bit to the most significant bit per errored symbol.
- `INC = 3`. When `mode` is set to `INC`, `bitmask` will be ignored. Instead, the bit error pattern initiates from 000000001, 000000010, 000000011, continuing up to 111111111, and repeating the sequence as 000000001…

2. `bitmask`: *hex2*, bit error mask for the errored symbols, big endian, only 9 bits are effective. The highest 3 bits are always ignored regardless of their values.

Example

```
# set
input:  0/1 PL1_CWE_BIT_ERR_MASK ROTATE_HIGH 0x0124
output: <OK>

# get
input:  0/1 PL1_CWE_BIT_ERR_MASK ?
output: 0/1 PL1_CWE_BIT_ERR_MASK ROTATE_HIGH 0x0124
```

PL1_CWE_CONTROL

`code:  445`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_CONTROL <action>

# get
<module-index>/<port-index> PL1_CWE_CONTROL ?
```

Description

Control the FEC codeword error injection

Actions

set, get

Parameters

1. `action`: *byte*, control action for FEC codeword error injection
   - `STOP = 0`
   - `START = 1`

Example

```
# set
input:  0/1 PL1_CWE_CONTROL START
output: <OK>

# get
input:  0/1 PL1_CWE_CONTROL ?
output: 0/1 PL1_CWE_CONTROL START
```

PL1_CWE_CYCLE

`code:  435`

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_CYCLE <loop> <cycle_len> <error_
↪len>

# get
<module-index>/<port-index> PL1_CWE_CYCLE ?
```

Description

Configure the FEC codeword error injection cycle.

Actions

set, get

Parameters

1. `loop`: *integer*, loop count of the FEC codeword error injection cycle. `<loop> == 0` means **continuous**.
2. `cycle_len`: *integer*, the number of FEC codewords in the cycle, **must be larger than 0 and multiple of 2**.
3. `error_len`: *integer*, the number of consecutive errored FEC codewords in a cycle, **must not be larger than** `cycle_len`.

Example

```
# set
input:  0/1 PL1_CWE_CYCLE 0 8 3
output: <OK>

# get
input:  0/1 PL1_CWE_CYCLE ?
output: 0/1 PL1_CWE_CYCLE 0 8 3
```

PL1_CWE_ERR_SYM_INDICES

`code:  436`

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_ERR_SYM_INDICES <error_sym_indices>
```

(continues on next page)

(continued from previous page)

```
# get
<module-index>/<port-index> PL1_CWE_ERR_SYM_INDICES ?
```

Description

Configure the positions of the errored symbols in errored codewords.

Actions

set, get

Parameters

1. `error_sym_indices`: *integer list*, the indices of the position of the errored symbols.
   - An empty list means there is no errored symbol in the errored codewords.
   - The indices in the list must not duplicate.
   - The indices in the list do not necessarily need to be sorted.
   - The maximum value of an index must not be larger than what the FEC schema allows, e.g. an index must not be larger than 543 for RS(544, 514).

Example

```
# set
input:  0/1 PL1_CWE_ERR_SYM_INDICES 0 10 23 54 87 500
output: <OK>

# get
input:  0/1 PL1_CWE_ERR_SYM_INDICES ?
output: 0/1 PL1_CWE_ERR_SYM_INDICES 0 10 23 54 87 500

# set
input:  0/1 PL1_CWE_ERR_SYM_INDICES
output: <OK>

# get
input:  0/1 PL1_CWE_ERR_SYM_INDICES ?
output: 0/1 PL1_CWE_ERR_SYM_INDICES
```

PL1_CWE_FEC_ENGINE

code: 438

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_FEC_ENGINE <engine_bitmask>

# get
<module-index>/<port-index> PL1_CWE_FEC_ENGINE ?
```

Description

Configure which FEC engines to use.

Actions

set, get

Parameters

1. `engine_bitmask`: *hex*, big endian.

- the highest bit corresponds to FEC engine 4 (0x08)
- the lowest bit corresponds to FEC engine 1 (0x01)

Example

```
# set
input:  0/1 PL1_CWE_FEC_ENGINE 0x09
output: <OK>

# get
input:  0/1 PL1_CWE_FEC_ENGINE ?
output: 0/1 PL1_CWE_FEC_ENGINE 0x09
```

PL1_CWE_FEC_STATS

code: 439

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_CWE_FEC_STATS ?
```

Description

FEC error injection statistics.

Actions

get

Parameters

1. `total_cw`: *long integer*, total codewords transmitted.
2. `total_correctable_cw`: *long integer*, total injected correctable codewords.
3. `total_uncorrectable_cw`: *long integer*, total uncorrectable codewords transmitted.
4. `total_error_free_cw`: *long integer*, total error-free codewords transmitted.
5. `total_symbol_error`: *long integer*, total injected symbol errors.
6. `total_bits`: *long integer*, total bits transmitted.
7. `total_bit_errors`: *long integer*, total injected bit errors.

Example

```
# get
input:  0/1 PL1_CWE_FEC_STATS ?
output: 0/1 PL1_CWE_FEC_STATS 80 10 0 70 50 43520 70
```

PL1_CWE_FEC_STATS_CLEAR

code: 446

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CWE_FEC_STATS_CLEAR
```

Description

Clear FEC codeword injection TX stats

Actions

set

Parameters

Example

```
# set
input:  0/1 PL1_CWE_FEC_STATS_CLEAR
output: <OK>
```

PP_ALARMS_ERRORS

code: 272

```
# get
<module-index>/<port-index> PP_ALARMS_ERRORS ?
```

Description

Obtain the error count of each alarm, PCS Error, FEC Error, Header Error, Align Error, BIP Error, and High BER Error.

Actions

get

Parameters

1. `total_alarms`: *integer*, the total alarm count.
2. `valid_mask`: *hex*, valid mask.
3. `los_error_cournt`: *long integer*, Loss of Signal alarms.
4. `total_pcs_error_count`: *long integer*, total PCS error alarms.
5. `total_fec_error_count`: *long integer*, total FEC error alarms.
6. `total_header_error_count`: *long integer*, total header error alarms.
7. `total_align_error_count`: *long integer*, total align error alarms.
8. `total_bip_error_count`: *long integer*, total BIP error alarms.
9. `total_highber_error_count`: *long integer*, total high BER error alarms.

Example

```
# get
input:  0/1 PP_ALARMS_ERRORS ?
output: 0/1 PP_ALARMS_ERRORS 1 0x57 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123
```

PP_FECMODE

code: 366

```
# set
<module-index>/<port-index> PP_FECMODE <mode>

# get
<module-index>/<port-index> PP_FECMODE ?
```

Description

FEC mode for port that supports FEC.

Actions

set, get

Parameters

1. `mode`: *byte*, FEC mode for port

- `OFF = 0`
- `FC_FEC = 3`
- `RS_FEC_KR = 4`
- `RS_FEC_KP = 5`
- `RS_FEC_INT = 6`

Example

```
# set
input:  0/1 PP_FECMODE RS_FEC_KR
output: <OK>

# get
input:  0/1 PP_FECMODE ?
output: 0/1 PP_FECMODE RS_FEC_KR
```

PP_PHYAUTONEG

`code:  362`

```
# set
<module-index>/<port-index> PP_PHYAUTONEG <fec_mode> <reserved_1>
 ↪<reserved_2> <reserved_3> <reserved_4>

# get
<module-index>/<port-index> PP_PHYAUTONEG ?
```

Description

Auto-negotiation settings of the PHY.

Actions

set, get

Parameters

1. `fec_mode`: *byte*, FEC mode ON or OFF
   - `OFF = 0`
   - `ON = 1`
2. `reserved_1`: *integer*, reserved for future use.
3. `reserved_2`: *integer*, reserved for future use.
4. `reserved_3`: *integer*, reserved for future use.
5. `reserved_4`: *integer*, reserved for future use.

Example

```
# set
input:  0/1 PP_PHYAUTONEG OFF 1 1 1 1
output: <OK>

# get
input:  0/1 PP_PHYAUTONEG ?
output: 0/1 PP_PHYAUTONEG OFF 1 1 1 1
```

PP_RXCLEAR

`code:  294`

```
# set
<module-index>/<port-index> PP_RXCLEAR
```

Description

Clear all the PCS/FEC RX Status counters.

Actions

set

Parameters

Example

```
# set
input:  0/1 PP_RXCLEAR
output: <OK>
```

PP_RXFECSTATS

code: 286

```
# get
<module-index>/<port-index> PP_RXFECSTATS ?
```

Description

Provides statistics on how many FEC codewords have been seen with a given number of symbol errors.

Actions

get

Parameters

1. `stats_type`: *long integer*, stats type (currently always 0).
2. `data_count`: *long integer*, the length of `correction_stats` + 1. Since `correction_stats` has a length of `n`+2, the value of `data_count` is `n`+3

3. `correction_stats`: *long integer list*, array of length `data_count-1`. The array shows how many FEC codewords have been seen with `[0, 1, 2, 3....n, >n]` symbol errors, where `n` is the uncorrectable error threshold, the point at which the error correction fails to recover because the symbol errors are too numerous.

Important:

- FEC codewords with <=n symbol errors are correctable.
- FEC codewords with >n symbol errors are uncorrectable.

---

4. `sum_of_zero_and_correctable_fec_codewords`: *long integer*, the sum of FEC codewords with <=n symbol errors.

Example

```
# get
input:  0/1 PP_RXFECSTATS ?
output: 0/1 PP_RXFECSTATS  0 18 11585906049992 460533565 106553 86 0 0␣
↪0 0 0 0 0 0 0 0 0 0 11586366690196
```

PP_RXLANEERRORS

`code:  271`

```
# get
<module-index>/<port-index> PP_RXLANEERRORS [<lane_index>] ?
```

Description

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

Actions

get

Parameters

1. `header_error_count`: *long integer*, the number of header errors
2. `alignment_error_count`: *long integer*, the number of alignment errors
3. `bip8_error_count`: *long integer*, the number of bip8 errors
4. `corrected_bit_errors`: *long integer*, corrected bit errors
5. `1_over_ber`: *long integer*, 1/BER value

Example

```
# get
input:  0/1 PP_RXLANEERRORS [0] ?
output: 0/1 PP_RXLANEERRORS [0] 0 0 0 29486878618 10070
```

PP_RXLANELOCK

`code: 290`

```
# get
<module-index>/<port-index> PP_RXLANELOCK [<lane_index>] ?
```

Description

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

Actions

get

Parameters

1. `headerlock`: *byte*, whether this lane has achieved header lock, and whether this lane has achieved alignment lock.

- `HEADEROFF = 0`
- `HEADERON = 1`
- `HEADEROFFUNSTABLE = 2`
- `HEADERONUNSTABLE = 3`

2. `alignlock`: *byte*, whether this lane has achieved header lock, and whether this lane has achieved alignment lock.

- `ALIGNOFF = 0`
- `ALIGNON = 1`
- `ALIGNOFFUNSTABLE = 2`
- `ALIGNONUNSTABLE = 3`

Example

```
# get
input:  0/1 PP_RXLANELOCK [0] ?
output: 0/1 PP_RXLANELOCK [0] HEADEROFF ALIGNOFF
```

PP_RXLANESTATUS

`code: 291`

```
# get
<module-index>/<port-index> PP_RXLANESTATUS [<lane_index>] ?
```

Description

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.

Actions

get

Parameters

1. `virtual_lane`: *integer*, the virtual lane index and actual skew for data received on a specified physical lane
2. `skew`: *integer*, the virtual lane index and actual skew for data received on a specified physical lane

Example

```
# get
input:  0/1 PP_RXLANESTATUS [0] ?
output: 0/1 PP_RXLANESTATUS [0] 1 1
```

PP_RXTOTALSTATS

code: 270

```
# get
<module-index>/<port-index> PP_RXTOTALSTATS ?
```

Description

Provides FEC Total counters.

Actions

get

Parameters

1. `total_rx_bits`: *long integer*, total RX bit count.
2. `total_rx_codewords`: *long integer*, total RX FEC codewords.
3. `total_corrected_codewords`: *long integer*, total corrected FEC codewords.
4. `total_uncorrectable_codewords`: *long integer*, total uncorrectable FEC codewords.
5. `total_corrected_symbols`: *long integer*, total corrected FEC symbols.
6. `total_pre_fec_ber`: *long integer*, total estimated pre-FEC BER.

   If no errors have been corrected, that is `total_corrected_codewords==0 && total_uncorrectable_codewords==0`, `total_pre_fec_ber = -total_rx_bits` simply outputs the number of error free bits divided by the BER confidence interval factor in a negative value to signal that this is an estimate. Then you can use `-1/total_pre_fec_ber` as the indicator of the upper bound of the BER.

   If errors are corrected, pre-FEC BER estimate is based on the symbols errors, assuming **one** error per symbol (max would be 10), `total_pre_fec_ber = total_rx_bits / (total_corrected_symbols + total_uncorrectable_codewords`

`* (max_fec_symbols+1))`. To get the real total pre-BER, calculate the inverse: `1/total_pre_fec_ber`.

7. `total_post_fec_ber`: *long integer*, total estimated post-FEC BER on uncorrected errors.

   If number of uncorrected errors is zero that is `total_uncorrectable_codewords == 0`, `total_post_fec_ber = - total_rx_bits` simply output the number of error free bits divided by the BER confidence interval factor in a negative value to signal that this is an estimate. Then you can use `-1/total_post_fec_ber` as the indicator of the upper bound of the BER.

   If number of uncorrected errors is not zero, `total_post_fec_ber = received_bits / (total_uncorrectable_codewords * (max_fec_symbols+1))`. To get the real total post-BER, calculate the inverse: `1/total_post_fec_ber`.

Example

```
# get
input:  0/1 PP_RXTOTALSTATS ?
output: 0/1 PP_RXTOTALSTATS  20723819061305600 3809525562740 2 54 23␣
↪23363944826725 23985901691325
```

PP_TXERRORRATE

`code:  283`

```
# set
<module-index>/<port-index> PP_TXERRORRATE <rate>

# get
<module-index>/<port-index> PP_TXERRORRATE ?
```

Description

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

Actions

set, get

Parameters

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

Example

```
# set
input:  0/1 PP_TXERRORRATE 1
output: <OK>

# get
input:  0/1 PP_TXERRORRATE ?
output: 0/1 PP_TXERRORRATE 1
```

PP_TXINJECTONE

code: 284

```
# set
<module-index>/<port-index> PP_TXINJECTONE
```

Description

Inject a single bit-level error into the SerDes where injection has been enabled.

Actions

set

Parameters

Example

```
# set
input:  0/1 PP_TXINJECTONE
output: <OK>
```

PP_TXLANECONFIG

code: 280

```
# set
<module-index>/<port-index> PP_TXLANECONFIG [<lane_index>] <virt_lane_
↪index> <skew>

# get
<module-index>/<port-index> PP_TXLANECONFIG [<lane_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `virt_lane_index`: *integer*, virtual lane index
2. `skew`: *integer*, the inserted skew on the lane, in bit units

Example

```
# set
input:  0/1 PP_TXLANECONFIG [0] 1 1
output: <OK>

# get
input:  0/1 PP_TXLANECONFIG [0] ?
output: 0/1 PP_TXLANECONFIG [0] 1 1
```

PP_TXLANEINJECT

code: 281

```
# set
<module-index>/<port-index> PP_TXLANEINJECT [<lane_index>] <inject_
↪error_type>
```

Description

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

Actions

set

Parameters

1. `inject_error_type`: *byte*, specifying what type of error to inject

- `HEADERERROR = 1`
- `ALIGNERROR = 2`
- `BIP8ERROR = 3`

Example

```
# set
input:  0/1 PP_TXLANEINJECT [0] HEADERERROR
output: <OK>
```

PRBS

Pseudo Random Binary Sequence

PP_PRBSTYPE

code: 378

```
# set
<module-index>/<port-index> PP_PRBSTYPE <prbs_inserted_type>
 ↪<polynomial> <invert> <statistics_mode>

# get
<module-index>/<port-index> PP_PRBSTYPE ?
```

Description

Defines the PRBS type used when the interface is in PRBS mode.

Actions

set, get

Parameters

1. *prbs_inserted_type* `: *byte*, specifying where the PRBS is inserted
   - CAUI_VIRTUAL = 0
   - PHY_LINE = 1
   - PHY_HOST = 2
   - TCVR = 3
2. polynomial: *byte*, specifying which PRBS that is used
   - PRBS7 = 0
   - PRBS9 = 1
   - PRBS11 = 2
   - PRBS15 = 3
   - PRBS23 = 4
   - PRBS31 = 5
   - PRBS58 = 6
   - PRBS49 = 7
   - PRBS10 = 8
   - PRBS20 = 9

- PRBS13 = 10
- SSPRQ = 24
- SQUARE_WAVE = 25

3. invert: *byte*, specifying if the PRBS is inverted

- NON_INVERTED = 0
- INVERTED = 1

4. statistics_mode: *byte*, specifying PRBS statistics mode, accumulative or for last second

- ACCUMULATIVE = 0
- PERSECOND = 1

Example

```
# set
input:  0/1 PP_PRBSTYPE CAUI_VIRTUAL PRBS7 NON_INVERTED ACCUMULATIVE
output: <OK>

# get
input:  0/1 PP_PRBSTYPE ?
output: 0/1 PP_PRBSTYPE CAUI_VIRTUAL PRBS7 NON_INVERTED ACCUMULATIVE
```

PP_RXPRBSSTATUS

code: 293

```
# get
<module-index>/<port-index> PP_RXPRBSSTATUS [<serdes_index>] ?
```

Description

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

Actions

get

Parameters

1. `byte_count`: *long integer*, the number of bytes received while in PRBS lock, the number of errors detected while in PRBS lock, and whether this SerDes is in PRBS lock.
2. `error_count`: *long integer*, the number of bytes received while in PRBS lock, the number of errors detected while in PRBS lock, and whether this SerDes is in PRBS lock.
3. `lock`: *byte*, the number of bytes received while in PRBS lock, the number of errors detected while in PRBS lock, and whether this SerDes is in PRBS lock.

- `PRBSOFF = 0`
- `PRBSON = 1`
- `PRBSOFFUNSTABLE = 2`
- `PRBSONUNSTABLE = 3`

Example

```
# get
input:  0/1 PP_RXPRBSSTATUS [0] ?
output: 0/1 PP_RXPRBSSTATUS [0] 123456789123 123456789123 PRBSOFF
```

PP_RXPRBSTYPE

`code:  365`

```
# set
<module-index>/<port-index> PP_RXPRBSTYPE <prbs_inserted_type> <prbs_
↪pattern> <invert> <statistics_mode>

# get
<module-index>/<port-index> PP_RXPRBSTYPE ?
```

Description

The RX PRBS type used when the interface is in PRBS mode.

Actions

set, get

Parameters

1. `prbs_inserted_type`: *byte*, PRBS inserted type
   - `CAUI_VIRTUAL = 0`
   - `PHY_LINE = 1`
   - `PHY_HOST = 2`
   - `TCVR = 3`
2. `prbs_pattern`: *byte*, PRBS pattern
   - `PRBS7 = 0`
   - `PRBS9 = 1`
   - `PRBS11 = 2`
   - `PRBS15 = 3`
   - `PRBS23 = 4`
   - `PRBS31 = 5`
   - `PRBS58 = 6`
   - `PRBS49 = 7`
   - `PRBS10 = 8`
   - `PRBS20 = 9`
   - `PRBS13 = 10`
3. `invert`: *byte*, PRBS invert state
   - `NON_INVERTED = 0`
   - `INVERTED = 1`
4. `statistics_mode`: *byte*, PRBS statistics mode
   - `ACCUMULATIVE = 0`
   - `PERSECOND = 1`

Example

```
# set
input:  0/1 PP_RXPRBSTYPE CAUI_VIRTUAL PRBS7 NON_INVERTED ACCUMULATIVE
output: <OK>

# get
input:  0/1 PP_RXPRBSTYPE ?
output: 0/1 PP_RXPRBSTYPE CAUI_VIRTUAL PRBS7 NON_INVERTED ACCUMULATIVE
```

PP_TXPRBSCONFIG

code: 282

```
# set
<module-index>/<port-index> PP_TXPRBSCONFIG [<serdes_index>] <prbs_
↪seed> <prbs_on_off> <error_on_off>

# get
<module-index>/<port-index> PP_TXPRBSCONFIG [<serdes_index>] ?
```

Description

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

Actions

set, get

Parameters

1. `prbs_seed`: *integer*, not used, set to 0.
2. `prbs_on_off`: *byte*, whether this SerDes is transmitting PRBS data.
   - `PRBSOFF = 0`
   - `PRBSON = 1`
3. `error_on_off`: *byte*, whether bit-level errors are injected into this SerDes
   - `ERRORSOFF = 0`
   - `ERRORSON = 1`

Example

```
# set
input:  0/1 PP_TXPRBSCONFIG [0] 1 PRBSOFF ERRORSOFF
output: <OK>

# get
input:  0/1 PP_TXPRBSCONFIG [0] ?
output: 0/1 PP_TXPRBSCONFIG [0] 1 PRBSOFF ERRORSOFF
```

PP_TXPRBSTYPE

code: 364

```
# set
<module-index>/<port-index> PP_TXPRBSTYPE <prbs_inserted_type> <prbs_
↪pattern> <invert>

# get
<module-index>/<port-index> PP_TXPRBSTYPE ?
```

Description

The TX PRBS type used when the interface is in PRBS mode.

Actions

set, get

Parameters

1. `prbs_inserted_type`: *byte*, PRBS inserted type
   - `CAUI_VIRTUAL = 0`
   - `PHY_LINE = 1`
   - `PHY_HOST = 2`
   - `TCVR = 3`
2. `prbs_pattern`: *byte*, PRBS pattern
   - `PRBS7 = 0`
   - `PRBS9 = 1`

- PRBS11 = 2
- PRBS15 = 3
- PRBS23 = 4
- PRBS31 = 5
- PRBS58 = 6
- PRBS49 = 7
- PRBS10 = 8
- PRBS20 = 9
- PRBS13 = 10

3. invert: *byte*, PRBS invert state

- NON_INVERTED = 0
- INVERTED = 1

Example

```
# set
input:  0/1 PP_TXPRBSTYPE PHY_LINE PRBS31 NON_INVERTED
output: <OK>

# get
input:  0/1 PP_TXPRBSTYPE ?
output: 0/1 PP_TXPRBSTYPE PHY_LINE PRBS31 NON_INVERTED
```

PMA

Physical Medium Attachment

PP_GRAYCODING

code: 421

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PP_GRAYCODING [<serdes_index>] <rx_mode>
↪<rx_endianness> <tx_mode> <tx_endianness>

# get
<module-index>/<port-index> PP_GRAYCODING [<serdes_index>] ?
```

Description

GET/SET Gray-Coding Configurations.

Actions

set, get

Parameters

1. `rx_mode`: *integer*, RX Mode Off/On
   - `OFF = 0`
   - `ON = 1`
2. `rx_endianness`: *integer*, RX Endianness Normal/Reverted(BigEndian/LittleEndian)
   - `NORMAL = 0`
   - `REVERTED = 1`
3. `tx_mode`: *integer*, TX Mode Off/On
   - `OFF = 0`
   - `ON = 1`
4. `tx_endianness`: *integer*, TX Endianness Normal/Reverted(BigEndian/LittleEndian)
   - `NORMAL = 0`
   - `REVERTED = 1`

Example

```
# set
input:  0/1 PP_GRAYCODING [0] OFF NORMAL OFF NORMAL
output: <OK>

# get
input:  0/1 PP_GRAYCODING [0] ?
output: 0/1 PP_GRAYCODING [0] OFF NORMAL OFF NORMAL
```

PP_LINKFLAP_ENABLE

code: 288

```
# set
<module-index>/<port-index> PP_LINKFLAP_ENABLE <on_off>

# get
<module-index>/<port-index> PP_LINKFLAP_ENABLE ?
```

Description

Enable / disable port 'link flap'.

Actions

set, get

Parameters

1. on_off: *byte*, whether link flap is enabled
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 PP_LINKFLAP_ENABLE OFF
output: <OK>

# get
input:  0/1 PP_LINKFLAP_ENABLE ?
output: 0/1 PP_LINKFLAP_ENABLE OFF
```

PP_LINKFLAP_PARAMS

code: 287

```
# set
<module-index>/<port-index> PP_LINKFLAP_PARAMS <duration> <period>
↪<repetition>

# get
<module-index>/<port-index> PP_LINKFLAP_PARAMS ?
```

Description

Set port 'link flap' parameters. Notice: Period must be larger than duration.

Actions

set, get

Parameters

1. `duration`: *integer*, 0 ms - 1000 ms; increments of 1 ms; 0 = permanently link down.
2. `period`: *integer*, 10 ms - 50000 ms; number of ms - must be multiple of 10 ms.
3. `repetition`: *integer*, 1 - 64K; 0 = continuous.

Example

```
# set
input:  0/1 PP_LINKFLAP_PARAMS 1 1 1
output: <OK>

# get
input:  0/1 PP_LINKFLAP_PARAMS ?
output: 0/1 PP_LINKFLAP_PARAMS 1 1 1
```

PP_PHYSETTINGS

code: 379

```
# set
<module-index>/<port-index> PP_PHYSETTINGS <link_training_on_off>
↪<precode_on_off> <graycode_on_off> <pam4_msb_lsb_swap>

# get
<module-index>/<port-index> PP_PHYSETTINGS ?
```

Description

Get and set low-level PHY settings.

Actions

set, get

Parameters

1. `link_training_on_off`: *byte*, enabling/disabling link training
   - `OFF = 0`
   - `ON = 1`
2. `precode_on_off`: *byte*, enabling/disabling link precode
   - `OFF = 0`
   - `ON = 1`
   - `DEFAULT = 2`
3. `graycode_on_off`: *byte*, enabling/disabling link graycode.

- OFF = 0
- ON = 1

4. pam4_msb_lsb_swap: *byte*, enabling/disabling PAM4 MSB/LSB swap.
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 PP_PHYSETTINGS OFF OFF OFF OFF
output: <OK>

# get
input:  0/1 PP_PHYSETTINGS ?
output: 0/1 PP_PHYSETTINGS OFF OFF OFF OFF
```

PP_PMAERRPUL_ENABLE

code: 300

```
# set
<module-index>/<port-index> PP_PMAERRPUL_ENABLE <on_off>

# get
<module-index>/<port-index> PP_PMAERRPUL_ENABLE ?
```

Description

Enable / disable 'PMA pulse error inject'.

Actions

set, get

Parameters

1. on_off: *byte*, whether PMA pulse error inject is enabled

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 PP_PMAERRPUL_ENABLE OFF
output: <OK>

# get
input:  0/1 PP_PMAERRPUL_ENABLE ?
output: 0/1 PP_PMAERRPUL_ENABLE OFF
```

PP_PMAERRPUL_PARAMS

code: 289

```
# set
<module-index>/<port-index> PP_PMAERRPUL_PARAMS <duration> <period>
↪<repetition> <coeff> <exp>

# get
<module-index>/<port-index> PP_PMAERRPUL_PARAMS ?
```

Description

The PMA pulse error inject.

Note: Period must be > duration. BER will be: coeff * 10exp

Actions

set, get

Parameters

1. `duration`: *integer*, 0 ms - 5000m s; increments of 1 ms; 0 = constant BER
2. `period`: *integer*, 10 ms - 50000 ms; number of ms - must be multiple of 10 ms
3. `repetition`: *integer*, 1 - 64K; 0 = continuous
4. `coeff`: *integer*, (0.01 < coeff < 9.99) * 100
5. `exp`: *integer*, -3 < exp < -17

Example

```
# set
input:  0/1 PP_PMAERRPUL_PARAMS 1 1 1 1 1
output: <OK>

# get
input:  0/1 PP_PMAERRPUL_PARAMS ?
output: 0/1 PP_PMAERRPUL_PARAMS 1 1 1 1 1
```

PP_PRECODING

`code:  420`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PP_PRECODING [<serdes_index>] <rx_mode>
↪<rx_endianness> <tx_mode> <tx_endianness>

# get
<module-index>/<port-index> PP_PRECODING [<serdes_index>] ?
```

Description

GET/SET Pre-Coding Configurations.

Actions

set, get

Parameters

1. `rx_mode`: *integer*, RX Mode Off/On/Auto
   - `OFF = 0`
   - `ON = 1`
2. `rx_endianness`: *integer*, RX Endianness Normal/Reverted(BigEndian/LittleEndian)
   - `NORMAL = 0`
   - `REVERTED = 1`
3. `tx_mode`: *integer*, TX Mode Off/On/Auto
   - `OFF = 0`
   - `ON = 1`
4. `tx_endianness`: *integer*, TX Endianness Normal/Reverted(BigEndian/LittleEndian)
   - `NORMAL = 0`
   - `REVERTED = 1`

Example

```
# set
input:  0/1 PP_PRECODING [0] OFF NORMAL OFF NORMAL
output: <OK>

# get
input:  0/1 PP_PRECODING [0] ?
output: 0/1 PP_PRECODING [0] OFF NORMAL OFF NORMAL
```

PP_PRECODINGSTATUS

code: 421

Attention: This command is deprecated.

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PP_PRECODINGSTATUS [<serdes_index>] ?
```

Description

GET Pre-Coding status

Actions

get

Parameters

1. `rx_mode`: *integer*, RX Mode Off/On
   - `OFF = 0`
   - `ON = 1`
2. `rx_endianness`: *integer*, RX Endianness Normal (Big Endian) / Reverted (Little Endian)
   - `NORMAL = 0`
   - `REVERTED = 1`
3. `tx_mode`: *integer*, TX Mode Off/On
   - `OFF = 0`
   - `ON = 1`
4. `tx_endianness`: *integer*, TX Endianness Normal (Big Endian) / Reverted (Little Endian)

- NORMAL = 0
- REVERTED = 1

Example

```
# get
input:  0/1 PP_PRECODING [0] ?
output: 0/1 PP_PRECODING [0] OFF NORMAL OFF NORMAL
```

ANLT

Auto-Negotiation and Link Training

P_AUTONEGSELECTION

code: 304

Attention: Only for the following modules:

- Odin-10G-5S-6P-CU

```
# set
<module-index>/<port-index> P_AUTONEGSELECTION <on_off>

# get
<module-index>/<port-index> P_AUTONEGSELECTION ?
```

Description

Whether the port responds to incoming auto-negotiation requests. Only applicable to electrical ports (RJ45).

Actions

set, get

Parameters

1. on_off: *byte*, whether the port responds to incoming auto-negotiation requests
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 P_AUTONEGSELECTION OFF
output: <OK>

# get
input:  0/1 P_AUTONEGSELECTION ?
output: 0/1 P_AUTONEGSELECTION OFF
```

PL1_ANLT

code: 441

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_ANLT <an_mode> <lt_mode>

# get
<module-index>/<port-index> PL1_ANLT ?
```

Description

ANLT action

Actions

set, get

Parameters

1. an_mode: *byte*, Autoneg mode
   - DISABLED = 0
   - ENABLED = 1
2. lt_mode: *byte*, Link Training mode
   - DISABLED = 0
   - ENABLED_AUTO = 1
   - ENABLED_INTERACTIVE = 2

Example

```
# set
input:  0/1 PL1_ANLT ENABLED ENABLED_AUTO
output: <OK>

# get
input:  0/1 PL1_ANLT ?
output: 0/1 PL1_ANLT ENABLED ENABLED_AUTO
```

PL1_AUTONEG_ABILITIES

code: 433

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_AUTONEG_ABILITIES ?
```

Description

Return the supported technology abilities, FEC abilities, and pause abilities by the port.

Actions

get

Parameters

1. `tech_abilities_supported`: *hex8*, supported technology abilities by the port. This returns a value in Hex of the format 0xHHHHHHHH (64 bits). Each bit corresponds to technology ability as shown below. A bit of 1 means the corresponding technology ability is supported by the port.

- bit 32-63: Reserved
- bit 31: Reserved
- bit 30: Reserved
- bit 29: 800G-ETC-CR8/KR8
- bit 28: 400G-ETC-CR8/KR8
- bit 27: 50G-ETC-CR2
- bit 26: 50G-ETC-KR2
- bit 25: 25G-ETC-CR
- bit 24: 25G-ETC-KR
- bit 23: 1.6TBASE-CR8/KR8
- bit 22: 800GBASE-CR4/KR4
- bit 21: 400GBASE-CR2/KR2
- bit 20: 200GBASE-CR1/KR1
- bit 19: 800GBASE-CR8/KR8
- bit 18: 400GBASE-CR4/KR4
- bit 17: 200GBASE-CR2/KR2
- bit 16: 100GBASE-CR1/KR1
- bit 15: 200GBASE-CR4/KR4
- bit 14: 100GBASE-CR2/KR2
- bit 13: 50GBASE-CR/KR
- bit 12: 5GBASE-KR

- bit 11: 2.5GBASE-KX
- bit 10: 25GBASE-CR/KR
- bit 9: 25GBASE-CR-S/KR-S
- bit 8: 100GBASE-CR4
- bit 7: 100GBASE-KR4
- bit 6: 100GBASE-KP4
- bit 5: 100GBASE-CR10
- bit 4: 40GBASE-CR4
- bit 3: 40GBASE-KR4
- bit 2: 10GBASE-KR
- bit 1: 10GBASE-KX4
- bit 0: 1000BASE-KX

2. `fec_modes_supported`: *hex*, supported FEC modes by the port. This returns a value in Hex of the format 0xH (8 bits). Each bit corresponds to FEC mode as shown below. A bit of 1 means the corresponding FEC mode is supported by the port.

- bit 7: Reserved
- bit 6: Reserved
- bit 5: Reserved
- bit 4: RS-FEC-Int
- bit 3: 25G FC-FEC Request
- bit 2: 25G RS-FEC Request
- bit 1: 10G FC-FEC Request
- bit 0: 10G FC-FEC Ability

3. `pause_modes_supported`: *hex*, pause abilities supported by the port. This returns a value in Hex of the format 0xH (8 bits). Each bit corresponds to pause mode as shown below. A bit of 1 means the corresponding FEC mode is supported by the port.

- bit 7: Reserved
- bit 6: Reserved
- bit 5: Reserved
- bit 4: Reserved
- bit 3: Reserved
- bit 2: Reserved
- bit 1: Asymmetric pause

- bit 0: Symmetric pause

Example

```
# get
input:  0/1 PL1_AUTONEG_ABILITIES ?
output: 0/1 PL1_AUTONEG_ABILITIES 0x0000000030000000 0x04 0x01
```

PL1_AUTONEG_CONFIG

`code: 440`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_AUTONEG_CONFIG <advertised_tech_
↪abilities> <advertised_fec_abilities> <advertised_pause_mode>

# get
<module-index>/<port-index> PL1_AUTONEG_CONFIG ?
```

Description

Auto-negotiation configuration for Freya

Actions

set, get

Parameters

1. `advertised_tech_abilities`: *hex8*, advertised technology abilities, 64-bit bitmask

- `IEEE_1000BASE_KX = 2^0 = 0x0000000000000001`
- `IEEE_10GBASE_KX4 = 2^1 = 0x0000000000000002`
- `IEEE_10GBASE_KR = 2^2 = 0x0000000000000004`

- IEEE_40GBASE_KR4 = 2^3 = 0x0000000000000008
- IEEE_40GBASE_CR4 = 2^4 = 0x0000000000000010
- IEEE_100GBASE_CR10 = 2^5 = 0x0000000000000020
- IEEE_100GBASE_KP4 = 2^6 = 0x0000000000000040
- IEEE_100GBASE_KR4 = 2^7 = 0x0000000000000080
- IEEE_100GBASE_CR4 = 2^8 = 0x0000000000000100
- IEEE_25GBASE_CR_S_KR_S = 2^9 = 0x0000000000000200
- IEEE_25GBASE_CR_KR = 2^10 = 0x0000000000000400
- IEEE_2DOT5GBASE_KX = 2^11 = 0x0000000000000800
- IEEE_5GBASE_KR = 2^12 = 0x0000000000001000
- IEEE_50GBASE_CR_KR = 2^13 = 0x0000000000002000
- IEEE_100GBASE_CR2_KR2 = 2^14 = 0x0000000000004000
- IEEE_200GBASE_CR4_KR4 = 2^15 = 0x0000000000008000
- IEEE_100GBASE_CR1_KR1 = 2^16 = 0x0000000000010000
- IEEE_200GBASE_CR2_KR2 = 2^17 = 0x0000000000020000
- IEEE_400GBASE_CR4_KR4 = 2^18 = 0x0000000000040000
- IEEE_800GBASE_CR8_KR8 = 2^19 = 0x0000000000080000
- IEEE_200GBASE_CR1_KR1 = 2^20 = 0x0000000000100000
- IEEE_400GBASE_CR2_KR2 = 2^21 = 0x0000000000200000
- IEEE_800GBASE_CR4_KR4 = 2^22 = 0x0000000000400000
- IEEE_1_6TBASE_CR8_KR8 = 2^23 = 0x0000000000800000
- EC_25GBASE_KR = 2^24 = 0x0000000001000000
- EC_25GBASE_CR = 2^25 = 0x0000000002000000
- EC_50GBASE_KR2 = 2^26 = 0x0000000004000000
- EC_50GBASE_CR2 = 2^27 = 0x0000000008000000
- EC_400GBASE_CR8_KR8 = 2^28 = 0x0000000010000000
- EC_800GBASE_CR8_KR8 = 2^29 = 0x0000000020000000

2. advertised_fec_abilities: *hex*, FEC capabilities to advertise, 8-bit bitmask

- 10G_FC_FEC_ABILITY= 2^0 = 0x01
- 10G_FC_FEC_REQUEST = 2^1 = 0x02
- 25G_RS_FEC_REQUEST = 2^2 = 0x04
- 25G_FC_FEC_REQUEST = 2^3 = 0x08

- RS_FEC_INT = 2^4 = 0x10

3. advertised_pause_mode: *hex*, pause ability to advertise, 8-bit bitmask

- SYM_PAUSE = 2^0 = 0x01
- ASYM_PAUSE = 2^1 = 0x02

Example

```
# set
input:  0/1 PL1_AUTONEG_CONFIG 0x0000000020080000 0x04 0x00
output: <OK>

# get
input:  0/1 PL1_AUTONEG_CONFIG ?
output: 0/1 PL1_AUTONEG_CONFIG 0x0000000020080000 0x04 0x00
```

PL1_AUTONEG_STATUS

code: 432

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_AUTONEG_STATUS ?
```

Description

Returns received technology abilities, FEC abilities, pause abilities, HCD technology ability, FEC mode result, and pause mode result.

Actions

get

Parameter

1. `mode:` : *integer*, the mode of autoneg
   - `DISABLED = 0`
   - `ENABLED = 1`
2. `autoneg_state`: *integer*, the state of auto-negotiation
   - `UNKNOWN = 0`
   - `ENABLE = 1`
   - `TRANSMIT_DISABLE = 2`
   - `ABILITY_DETECT = 3`
   - `ACKNOWLEDGE_DETECT = 4`
   - `COMPLETE_ACKNOWLEDGE = 5`
   - `NEXT_PAGE_WAIT = 6`
   - `AN_GOOD_CHECK = 7`
   - `AN_GOOD = 8`
3. `received_tech_abilities`: *hex8*, received technology abilities, 64-bit bitmask
   - `IEEE_1000BASE_KX = 2^0 = 0x0000000000000001`
   - `IEEE_10GBASE_KX4 = 2^1 = 0x0000000000000002`
   - `IEEE_10GBASE_KR = 2^2 = 0x0000000000000004`
   - `IEEE_40GBASE_KR4 = 2^3 = 0x0000000000000008`
   - `IEEE_40GBASE_CR4 = 2^4 = 0x0000000000000010`
   - `IEEE_100GBASE_CR10 = 2^5 = 0x0000000000000020`
   - `IEEE_100GBASE_KP4 = 2^6 = 0x0000000000000040`
   - `IEEE_100GBASE_KR4 = 2^7 = 0x0000000000000080`
   - `IEEE_100GBASE_CR4 = 2^8 = 0x0000000000000100`
   - `IEEE_25GBASE_CR_S_KR_S = 2^9 = 0x0000000000000200`
   - `IEEE_25GBASE_CR_KR = 2^10 = 0x0000000000000400`
   - `IEEE_2DOT5GBASE_KX = 2^11 = 0x0000000000000800`
   - `IEEE_5GBASE_KR = 2^12 = 0x0000000000001000`

- IEEE_50GBASE_CR_KR = 2^13 = 0x0000000000002000
- IEEE_100GBASE_CR2_KR2 = 2^14 = 0x0000000000004000
- IEEE_200GBASE_CR4_KR4 = 2^15 = 0x0000000000008000
- IEEE_100GBASE_CR1_KR1 = 2^16 = 0x0000000000010000
- IEEE_200GBASE_CR2_KR2 = 2^17 = 0x0000000000020000
- IEEE_400GBASE_CR4_KR4 = 2^18 = 0x0000000000040000
- IEEE_800GBASE_CR8_KR8 = 2^19 = 0x0000000000080000
- IEEE_200GBASE_CR1_KR1 = 2^20 = 0x0000000000100000
- IEEE_400GBASE_CR2_KR2 = 2^21 = 0x0000000000200000
- IEEE_800GBASE_CR4_KR4 = 2^22 = 0x0000000000400000
- IEEE_1_6TBASE_CR8_KR8 = 2^23 = 0x0000000000800000
- EC_25GBASE_KR = 2^24 = 0x0000000001000000
- EC_25GBASE_CR = 2^25 = 0x0000000002000000
- EC_50GBASE_KR2 = 2^26 = 0x0000000004000000
- EC_50GBASE_CR2 = 2^27 = 0x0000000008000000
- EC_400GBASE_CR8_KR8 = 2^28 = 0x0000000010000000
- EC_800GBASE_CR8_KR8 = 2^29 = 0x0000000020000000

4. `received_fec_abilities`: *hex*, received FEC capabilities, 8-bit bitmask

- 10G_FC_FEC_ABILITY = 2^0 = 0x01
- 10G_FC_FEC_REQUEST = 2^1 = 0x02
- 25G_RS_FEC_REQUEST = 2^2 = 0x04
- 25G_FC_FEC_REQUEST = 2^3 = 0x08
- RS_FEC_INT = 2^4 = 0x10

5. `received_pause_mode`: *hex*, received pause mode, 8-bit bitmask.

- SYM_PAUSE = 2^0 = 0x01
- ASYM_PAUSE = 2^1 = 0x02

6. `tech_ability_hcd_status`: *integer*, whether technology ability highest common denominator is found.

- SUCCESS = 1
- FAILED = 2

7. `tech_ability_hcd_value`: *integer*, technology ability highest common denominator.

- IEEE_1000BASE_KX = 0

- IEEE_10GBASE_KX4 = 1
- IEEE_10GBASE_KR = 2
- IEEE_40GBASE_KR4 = 3
- IEEE_40GBASE_CR4 = 4
- IEEE_100GBASE_CR10 = 5
- IEEE_100GBASE_KP4 = 6
- IEEE_100GBASE_KR4 = 7
- IEEE_100GBASE_CR4 = 8
- IEEE_25GBASE_CR_S_KR_S = 9
- IEEE_25GBASE_CR_KR = 10
- IEEE_2DOT5GBASE_KX = 11
- IEEE_5GBASE_KR = 12
- IEEE_50GBASE_CR_KR = 13
- IEEE_100GBASE_CR2_KR2 = 14
- IEEE_200GBASE_CR4_KR4 = 15
- IEEE_100GBASE_CR1_KR1 = 16
- IEEE_200GBASE_CR2_KR2 = 17
- IEEE_400GBASE_CR4_KR4 = 18
- IEEE_800GBASE_CR8_KR8 = 19
- IEEE_200GBASE_CR1_KR1 = 20
- IEEE_400GBASE_CR2_KR2 = 21
- IEEE_800GBASE_CR4_KR4 = 22
- IEEE_1_6TBASE_CR8_KR8 = 23
- EC_25GBASE_KR = 24
- EC_25GBASE_CR = 25
- EC_50GBASE_KR2 = 26
- EC_50GBASE_CR2 = 27
- EC_400GBASE_CR8_KR8 = 28
- EC_800GBASE_CR8_KR8 = 29

Note: The value is the bit position of bitmask (big endian).

8. `fec_mode_result`: *integer*, FEC mode result from autoneg.

- OFF = 0
- FC_FEC = 3
- RS_FEC_KR = 4
- RS_FEC_KP = 5
- RS_FEC_INT = 6

9. pause_mode_result: *integer*, pause mode result from autoneg.

- NO_PAUSE = 0
- SYM_PAUSE = 1
- ASYM_PAUSE = 2

Example

```
# get
input:  0/1 PL1_AUTONEG_STATUS ?
output: 0/1 PL1_AUTONEG_STATUS ENABLED AN_GOOD 0x0000000020080000 0x04␣
↪NO_PAUSE SUCCESS IEEE_800GBASE_CR8_KR8 RS_FEC_KP NO_PAUSE
```

PL1_AUTONEGINFO

code: 385

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_AUTONEGINFO [<page>] ?
```

Description

Get L1 auto-negotiation information. Information is split into a number of pages.

Actions

get

Parameters

1. `rx_link_codeword_count`: *integer*, received number of Link Code Words (Base Pages).
2. `rx_next_page_message_count`: *integer*, received number of Next Pages - Message Pages.
3. `rx_next_page_unformatted_count`: *integer*, received number of Next Pages - Unformatted Pages.
4. `tx_link_codeword_count`: *integer*, transmitted number of Link Code Words (Base Pages).
5. `tx_next_page_message_count`: *integer*, transmitted number of Next Pages - Message Pages.
6. `tx_next_page_unformatted_count`: *integer*, transmitted number of Nex Pages - Unformatted Pages.
7. `negotiation_hcd_fail_count`: *integer*, number of negotiation HCD (Highest Common Denominator) failures.
8. `negotiation_fec_fail_count`: *integer*, number of negotiation FEC failures.
9. `negotiation_loss_of_sync_count`: *integer*, number of negotiation Loss of Sync failures.
10. `negotiation_timeout_count`: *integer*, number of negotiation timeouts.
11. `negotiation_success_count`: *integer*, number of negotiation successes.
12. `duration_us`: *integer*, duration of the auto-negotiation in microseconds, from autoneg is enabled on the port to the negotiation is finished.

Example

```
# get
input:  0/1 PL1_AUTONEGINFO [0x00] ?
output: 0/1 PL1_AUTONEGINFO [0x00] 1 1 1 1 1 1 0 0 0 0 1 0
```

PL1_LINKTRAIN_CMD

code: 389

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_LINKTRAIN_CMD [<serdes>] <cmd> <arg>

# get
<module-index>/<port-index> PL1_LINKTRAIN_CMD [<serdes>] ?
```

Description

Link training RPC. Issue link training commands on a given serdes and poll for status.

The parameters returned by the get method is <cmd> <arg> <result> <flag>

Actions

set, get

Parameters

1. cmd: *byte*, the Link Training command code.
   - CMD_NOP = 0, No operation. Used for ping testing
   - CMD_INC = 1, Increment the coeff provided in ARG
   - CMD_DEC = 2, Decrement the coeff provided in ARG
   - CMD_PRESET = 3, Set the preset provided in ARG
   - CMD_ENCODING = 4, Set encoding provided in ARG
   - CMD_LOCAL_TRAINED = 255, Signal training completed
2. arg: *byte*, the arguments for the corresponding LT command.

   For <cmd> being CMD_INC and CMD_DEC:

   - PRE1 = 0, Pre1 coeff c(-1)
   - MAIN = 1, Pre1 coeff c(0)

- `POST = 2`, Pre1 coeff c(1)
- `PRE2 = 3`, Pre1 coeff c(-2)
- `PRE3 = 4`, Pre1 coeff c(-3)

For <cmd> being `CMD_PRESET`:

- `PRESET_1 = 0`, preset 1
- `PRESET_2 = 1`, preset 2
- `PRESET_3 = 2`, preset 3
- `PRESET_4 = 3`, preset 4
- `PRESET_5 = 4`, preset 5

For <cmd> being `CMD_ENCODING`:

- `PAM2 = 0`, NRZ
- `PAM4 = 1`, PAM4
- `PAM4_WITH_PRECODING = 2`, PAM4 with precoding

3. `result`: *byte*, the LT command result.

- `UNKNOWN = 0x00 | 0`, Unknown result
- `SUCCESS = 0x00 | 1`, Command successfully completed
- `TIMEOUT = 0x00 | 2`, Command timeout
- `FAILED = 0x00 | 3`, Command failed
- `COEFF_STS_NOT_UPDATED = 0x80 | 0`, Coefficient did not update
- `COEFF_STS_UPDATED = 0x80 | 1`, Coefficient updated
- `COEFF_STS_AT_LIMIT = 0x80 | 2`, Coefficient at limit
- `COEFF_STS_NOT_SUPPORTED = 0x80 | 3`, Coefficient not supported
- `COEFF_STS_EQ_LIMIT = 0x80 | 4`, EQ limit reached
- `COEFF_STS_C_AND_EQ_LIMIT = 0x80 | 6`, Coefficient and EQ limit reached

4. `flag`: *byte*, the LT command flag bitmask.

- `NEW = 1`, New command
- `IN_PROGRESS = 2`, Command in progress
- `DONE = 4`, Command done
- `LOCK = 8`, Link locked
- `LOCK_LOST = 16`, Link lock lost
- `OVERRUN = 32`, Overrun detected

Example

```
# set
input:  0/0 PL1_LINKTRAIN_CMD [0x00] CMD_INC PRE1
output: <OK>

# get
input:  0/0 PL1_LINKTRAIN_CMD [0x00] ?
output: 0/0 PL1_LINKTRAIN_CMD [0x00] CMD_INC PRE1 COEFF_STS_UPDATED␣
↪DONE

---

# set
input:  0/0 PL1_LINKTRAIN_CMD [0x00] CMD_DEC MAIN
output: <OK>

# get
input:  0/0 PL1_LINKTRAIN_CMD [0x00] ?
output: 0/0 PL1_LINKTRAIN_CMD [0x00] CMD_DEC MAIN COEFF_STS_UPDATED␣
↪DONE

---

# set
input:  0/0 PL1_LINKTRAIN_CMD [0x00] CMD_PRESET PRESET_2
output: <OK>

# get
input:  0/0 PL1_LINKTRAIN_CMD [0x00] ?
output: 0/0 PL1_LINKTRAIN_CMD [0x00] CMD_PRESET PRESET_2 SUCCESS DONE

---

# set
input:  0/0 PL1_LINKTRAIN_CMD [0x00] CMD_ENCODING PAM4_WITH_PRECODING
output: <OK>

# get
input:  0/0 PL1_LINKTRAIN_CMD [0x00] ?
output: 0/0 PL1_LINKTRAIN_CMD [0x00] CMD_ENCODING PAM4_WITH_PRECODING␣
↪SUCCESS DONE

---

# set
input:  0/0 PL1_LINKTRAIN_CMD [0x00] CMD_LOCAL_TRAINED 0
```

(continues on next page)

(continued from previous page)

```
output: <OK>

# get
input:  0/0 PL1_LINKTRAIN_CMD [0x00] ?
output: 0/0 PL1_LINKTRAIN_CMD [0x00] CMD_LOCAL_TRAINED 0 SUCCESS DONE
```

PL1_LINKTRAIN_CONFIG

code: 443

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_LINKTRAIN_CONFIG <oos_preset> <timeout_
↪mode>

# get
<module-index>/<port-index> PL1_LINKTRAIN_CONFIG ?
```

Description

Per-port link training settings

Actions

set, get

Parameters

1. `oos_preset`: *byte*, out-of-sync preset mode. If `<oos_preset> == 0`, the out-of-sync preset will use the values defined by IEEE spec, which is actually preset 1. If `<oos_preset> == 1`, the out-of-sync preset will use whatever the current tap values of the serdes.

- `IEEE = 0`
- `CURRENT = 1`

2. `timeout_mode`: *byte*, timeout mode

- DEFAULT = 0
- DISABLED = 255

Example

```
# set
input:  0/1 PP_LINKTRAIN IEEE DEFAULT
output: <OK>

# get
input:  0/1 PP_LINKTRAIN ?
output: 0/1 PP_LINKTRAIN IEEE DEFAULT
```

PL1_LINKTRAIN_STATUS

code: 444

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_LINKTRAIN_STATUS [<serdes_index>] ?
```

Description

Per-lane link training status

Actions

get

Parameters

1. `mode`: *byte*, link training mode of a lane of a port.
   - `DISABLED = 0`
   - `ENABLED = 1`
2. `status`: *byte*, lane status of a lane of a port.
   - `NOT_TRAINED = 0`
   - `TRAINED = 1`
3. `failure`: *byte*, failure type of the lane.
   - `NO_FAILURE = 0`
   - `FRAME_LOCK_ERROR = 1`
   - `SNR_BELOW_THRESHOLD = 2`
   - `TIME_OUT_FAILURE = 3`

Example

```
# get
input:  0/1 PL1_LINKTRAIN_STATUS [0] ?
output: 0/1 PL1_LINKTRAIN_STATUS [0] ENABLED TRAINED NO_FAILURE
```

PL1_LINKTRAININFO

`code:  386`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_LINKTRAININFO [<serdes>, <page>] ?
```

Description

Get L1 link training information. Information is per Serdes and split into a number of pages.

Actions

get

Parameters

1. `duration_us`: *integer*, duration of the link training process in microseconds, from autoneg is enabled on the port to the negotiation is finished.
2. `lock_lost_count`: *integer*, number of lost locks.
3. `pre1_current_level`: *integer*, c(-1) current local value.
4. `pre1_rx_increment_req_count`: *integer*, c(-1) received number of increment requests from link partner.
5. `pre1_rx_decrement_req_count`: *integer*, c(-1) received number of decrement requests from link partner.
6. `pre1_rx_coeff_eq_limit_reached_count`: *integer*, c(-1) received number of equalization coefficient request limits reached from link partner.
7. `pre1_rx_eq_limit_reached_count`: *integer*, c(-1) received number of equalization request limits reached from link partner.
8. `pre1_rx_coeff_not_supported_count`: *integer*, c(-1) received number of coefficients not supported from link partner.
9. `pre1_rx_coeff_at_limit_count`: *integer*, c(-1) received number of coefficients at limit from link partner.
10. `pre1_tx_increment_req_count`: *integer*, c(-1) transmitted number of increment requests to link partner.
11. `pre1_tx_decrement_req_count`: *integer*,c(-1) transmitted number of decrement requests to link partner.
12. `pre1_tx_coeff_eq_limit_reached_count`: *integer*, c(-1) transmitted number of equalization coefficient request limits reached to link partner.
13. `pre1_tx_eq_limit_reached_count`: *integer*, c(-1) transmitted number of equalization request limits reached to link partner.
14. `pre1_tx_coeff_not_supported_count`: *integer*, c(-1) transmitted number of coefficients not supported to link partner.
15. `pre1_tx_coeff_at_limit_count`: *integer*, c(-1) transmitted number of coefficients at limit to link partner.

16. `main_current_level`: *integer*, c(0) current local value.
17. `main_rx_increment_req_count`: *integer*, c(0) received number of increment requests from link partner.
18. `main_rx_decrement_req_count`: *integer*, c(0) received number of decrement requests from link partner.
19. `main_rx_coeff_eq_limit_reached_count`: *integer*, c(0) received number of equalization coefficient request limits reached from link partner.
20. `main_rx_eq_limit_reached_count`: *integer*, c(0) received number of equalization request limits reached from link partner.
21. `main_rx_coeff_not_supported_count`: *integer*, c(0) received number of coefficients not supported from link partner.
22. `main_rx_coeff_at_limit_count`: *integer*, c(0) received number of coefficients at limit from link partner.
23. `main_tx_increment_req_count`: *integer*, c(0) transmitted number of increment requests to link partner.
24. `main_tx_decrement_req_count`: *integer*,c(0) transmitted number of decrement requests to link partner.
25. `main_tx_coeff_eq_limit_reached_count`: *integer*, c(0) transmitted number of equalization coefficient request limits reached to link partner.
26. `main_tx_eq_limit_reached_count`: *integer*, c(0) transmitted number of equalization request limits reached to link partner.
27. `main_tx_coeff_not_supported_count`: *integer*, c(0) transmitted number of coefficients not supported to link partner.
28. `main_tx_coeff_at_limit_count`: *integer*, c(0) transmitted number of coefficients at limit to link partner.
29. `post1_current_level`: *integer*, c(1) current local value.
30. `post1_rx_increment_req_count`: *integer*, c(1) received number of increment requests from link partner.
31. `post1_rx_decrement_req_count`: *integer*, c(1) received number of decrement requests from link partner.
32. `post1_rx_coeff_eq_limit_reached_count`: *integer*, c(1) received number of equalization coefficient request limits reached from link partner.
33. `post1_rx_eq_limit_reached_count`: *integer*, c(1) received number of equalization request limits reached from link partner.
34. `post1_rx_coeff_not_supported_count`: *integer*, c(1) received number of coefficients not supported from link partner.
35. `post1_rx_coeff_at_limit_count`: *integer*, c(1) received number of coefficients at limit from link partner.

36. `post1_tx_increment_req_count`: *integer*, c(1) transmitted number of increment requests to link partner.
37. `post1_tx_decrement_req_count`: *integer*,c(1) transmitted number of decrement requests to link partner.
38. `post1_tx_coeff_eq_limit_reached_count`: *integer*, c(1) transmitted number of equalization coefficient request limits reached to link partner.
39. `post1_tx_eq_limit_reached_count`: *integer*, c(1) transmitted number of equalization request limits reached to link partner.
40. `post1_tx_coeff_not_supported_count`: *integer*, c(1) transmitted number of coefficients not supported to link partner.
41. `post1_tx_coeff_at_limit_count`: *integer*, c(1) transmitted number of coefficients at limit to link partner.
42. `pre2_current_level`: *integer*, c(-2) current local value.
43. `pre2_rx_increment_req_count`: *integer*, c(-2) received number of increment requests from link partner.
44. `pre2_rx_decrement_req_count`: *integer*, c(-2) received number of decrement requests from link partner.
45. `pre2_rx_coeff_eq_limit_reached_count`: *integer*, c(-2) received number of equalization coefficient request limits reached from link partner.
46. `pre2_rx_eq_limit_reached_count`: *integer*, c(-2) received number of equalization request limits reached from link partner.
47. `pre2_rx_coeff_not_supported_count`: *integer*, c(-2) received number of coefficients not supported from link partner.
48. `pre2_rx_coeff_at_limit_count`: *integer*, c(-2) received number of coefficients at limit from link partner.
49. `pre2_tx_increment_req_count`: *integer*, c(-2) transmitted number of increment requests to link partner.
50. `pre2_tx_decrement_req_count`: *integer*, c(-2) transmitted number of decrement requests to link partner.
51. `pre2_tx_coeff_eq_limit_reached_count`: *integer*, c(-2) transmitted number of equalization coefficient request limits reached to link partner.
52. `pre2_tx_eq_limit_reached_count`: *integer*, c(-2) transmitted number of equalization request limits reached to link partner.
53. `pre2_tx_coeff_not_supported_count`: *integer*, c(-2) transmitted number of coefficients not supported to link partner.
54. `pre2_tx_coeff_at_limit_count`: *integer*, c(-2) transmitted number of coefficients at limit to link partner.
55. `pre3_current_level`: *integer*, c(-3) current local value.

56. `pre3_rx_increment_req_count`: *integer*, c(-3) received number of increment requests from link partner.
57. `pre3_rx_decrement_req_count`: *integer*, c(-3) received number of decrement requests from link partner.
58. `pre3_rx_coeff_eq_limit_reached_count`: *integer*, c(-3) received number of equalization coefficient request limits reached from link partner.
59. `pre3_rx_eq_limit_reached_count`: *integer*, c(-3) received number of equalization request limits reached from link partner.
60. `pre3_rx_coeff_not_supported_count`: *integer*, c(-3) received number of coefficients not supported from link partner.
61. `pre3_rx_coeff_at_limit_count`: *integer*, c(-3) received number of coefficients at limit from link partner.
62. `pre3_tx_increment_req_count`: *integer*, c(-3) transmitted number of increment requests to link partner.
63. `pre3_tx_decrement_req_count`: *integer*,c(-3) transmitted number of decrement requests to link partner.
64. `pre3_tx_coeff_eq_limit_reached_count`: *integer*, c(-3) transmitted number of equalization coefficient request limits reached to link partner.
65. `pre3_tx_eq_limit_reached_count`: *integer*, c(-3) transmitted number of equalization request limits reached to link partner.
66. `pre3_tx_coeff_not_supported_count`: *integer*, c(-3) transmitted number of coefficients not supported to link partner.
67. `pre3_tx_coeff_at_limit_count`: *integer*, c(-3) transmitted number of coefficients at limit to link partner.
68. `prbs_total_bits_high`: *integer*, PRBS total bits (most significant 32-bit).
69. `prbs_total_bits_low`: *integer*, PRBS total bits (least significant 32-bit).
70. `prbs_total_error_bits_high`: *integer*, PRBS total error bits (most significant 32-bit, only bit 15-0 should be used).
71. `prbs_total_error_bits_low`: *integer*, PRBS total error bits (least significant 32-bit).
72. `frame_lock`: *integer*, frame lock status.
    - `LOST = 0`
    - `LOCKED = 1`
73. `remote_frame_lock`: *integer*, frame lock status of the remote end.
    - `LOST = 0`
    - `LOCKED = 1`
74. `num_frame_errors`: *integer*, number of frame errors received.

75. num_overruns: *integer*, number of overruns
76. num_last_ic_received: *integer*, the last preset request received
77. num_last_ic_sent: *integer*, the last preset request sent

Example

```
# get
input:  0/0 PL1_LINKTRAININFO [0, 0] ?
output: 0/0 PL1_LINKTRAININFO [0, 0] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 LOCKED LOCKED 0 0 1 1
```

PL1_LOG

code: 387

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PL1_LOG ?
```

Description

Return a log line of ANLT of all serdes on a port (latest 100 lines)

Actions

get

Parameters

1. `log_string`: *string*, return a log line from *AN*/*LT* for the given Serdes.

Example

```
# get
input:  0/0 PL1_LOG ?
```

PL1_CFG_TMP

`code:  388`

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CFG_TMP [<serdes>, <type>] <values>

# get
<module-index>/<port-index> PL1_CFG_TMP [<serdes>, <type>] ?
```

Description

This command performs various actions based on the <type> value.

Actions

set, get

Indices

`type`: *integer*, Layer-1 configuration type. Different types have different interpretations of `value`

- `0 (ANLT Auto Link Recovery)`, *AN*/*LT* auto link recovery
- `1 (Auto-Negotiation Loopback)`, Auto-negotiation loopback config
- `2 (Link Training Initial Modulation)`, Initial modulation of link training
- `3 (Low-Level Debug Info)`, Low-level debug, used to initialize low-level debug, get-only.
- `4 (Link Training Algorithm)`, For link training algorithm selection
- `5 (ANLT Log Control)`, control what should be logged by ANLT
- `6 (ANLT Strict Mode)`, set *AN*/*LT* strict mode. In strict mode, errored framed will be ignored
- `7 (ANLT XLA Mode)`, set *XLA* mode. If enabled XLA dumps will, if triggered, be logged automatically

Parameters

1. `values`: *integer list*, the meaning of <value> varies depending on the <type>.

PL1_CFG_TMP [0, 0] <int>

This command manages the auto-restart features. The <int> parameter represents the sum of the following values:

- 1: Enable AN+LT auto-restart when a **link down** condition is detected. A "link down" state signifies the loss of a valid input signal, which can occur due to events such as cable unplugging and re-plugging, TX disable, or link flap on the link partner's end. The auto-restart process will continue until the link is re-established. Please note that this setting is only effective when AN and/or LT are enabled.
- 2: If LT is enabled and experiences a failure on either side, the port will initiate the AN+LT restart process repeatedly until LT succeeds. This functionality is only applicable when LT is enabled.
- The default is 0, indicating that all AN+LT auto-restart options are disabled by default.

Example

```
PL1_CFG_TMP [0, 0] 3

Enable both the described restart features.
```

PL1_CFG_TMP [0, 1] <int>

This command controls whether the port should permit loopback during *AN* operations. The <int> parameter has the following values:

- 0: Not allow AN in loopback. The port must be connected to another port. (default)
- 1: Allow AN in loopback. The port can loop back to itself.

Example

```
PL1_CFG_TMP [0, 1] 1

Allow port in loopback when exercising auto-negotiation.
```

PL1_CFG_TMP [<serdes>, 2] <int>

This command controls the initial modulation of Link Training. The <int> parameter has the following values:

- NRZ = 0: link training encoding NRZ.
- PAM4 = 1: link training encoding PAM4.
- PAM4_WITH_PRECODING = 2: link training encoding PAM4 with Precoding.

Example

```
PL1_CFG_TMP [0, 2] 0

Set the initial modulation of serdes lane 0 to NRZ.
```

PL1_CFG_TMP [<serdes>, 3] ?

Initializes the communication parameters required to read the configuration of a Serializer/Deserializer. It takes in a port object used for communication, and the index of the Serializer/Deserializer to read (serdes). The function returns an object of type AnLtLowLevelInfo, which contains low-level communication information.

Returns a list of integers that parsed into the following structure:

```
class AnLtLowLevelInfo:
    base: int
    rx_gtm_base: int
    rx_serdes: int
    tx_gtm_base: int
    tx_serdes: int
```

Example

```
PL1_CFG_TMP [0, 3] ?
```

PL1_CFG_TMP [<serdes>, 4] <int>

This command controls the link training algorithm used by the port. The <int> parameter has the following values:

- INTERACTIVE = 0: Interactive link training (user manually does link training)
- ALG0 = 1: Algorithm 0 (Xena proprietary algorithm)
- ALGN1 = 2: Algorithm -1 (Xena proprietary algorithm)

Example

```
PL1_CFG_TMP [0, 4] 1

Set the LT algorithm to Algorithm 0.
```

PL1_CFG_TMP [<serdes>, 5] <[int]>

This command controls the ANLT log output from the server. To activate specific log output options, include the corresponding values in the <[int]> list of integers.

- 2: debug log output (0x02)
- 4: auto-negotiation trace output (0x04)
- 8: link training trace output (0x08)
- 16: link training algorithm trace (0x10)
- 131072: auto-negotiation state machine transitions (0x20000)
- 262144: auto-negotiation stimuli state machine transitions (0x40000)
- 524288: link training state machine transitions (0x80000)
- 1048576: link training coefficient state machine transitions (0x100000)
- 2097152: link training stimuli state machine transitions (0x200000)
- 4194304: link training algorithm 0 state machine transitions (0x400000)
- 8388608: link training algorithm -1 state machine transitions (0x800000)

Example

```
PL1_CFG_TMP [0, 5] 2 4 8

Set serdes lane 0 to output debug log output, auto-negotiation trace␣
↪output, and link training trace output.
```

PL1_CFG_TMP [<serdes>, 6] <int>

This command controls the ANLT strict mode. In strict mode, errored framed will be ignored. The <int> parameter has the following values:

- 0: disable on serdes lane
- 1: enable ANLT strict mode on serdes lane

Example

```
PL1_CFG_TMP [0, 6] 1

enable ANLT strict mode on serdes lane 0.
```

PP_AUTONEG

code: 381

Attention: Only for the following modules:

- Thor-400G-7S-1P

```
# set
<module-index>/<port-index> PP_AUTONEG <mode> <tec_ability> <fec_
↪capable> <fec_requested> <pause_mode>

# get
<module-index>/<port-index> PP_AUTONEG ?
```

Description

Auto-negotiation configurations

Actions

set, get

Parameters

1. mode: *integer*, auto-negotiation on/off status
   - ANEG_OFF = 0
   - ANEG_ON = 1
2. tec_ability: *integer*, technology ability to advertise
   - DEFAULT_TECH_MODE = 0
   - IEEE_1000BASE_KX = 2^0 = 0x0000000000000001
   - IEEE_10GBASE_KX4 = 2^1 = 0x0000000000000002
   - IEEE_10GBASE_KR = 2^2 = 0x0000000000000004

- IEEE_40GBASE_KR4 = 2^3 = 0x0000000000000008
- IEEE_40GBASE_CR4 = 2^4 = 0x0000000000000010
- IEEE_100GBASE_CR10 = 2^5 = 0x0000000000000020
- IEEE_100GBASE_KP4 = 2^6 = 0x0000000000000040
- IEEE_100GBASE_KR4 = 2^7 = 0x0000000000000080
- IEEE_100GBASE_CR4 = 2^8 = 0x0000000000000100
- IEEE_25GBASE_CR_S_KR_S = 2^9 = 0x0000000000000200
- IEEE_25GBASE_CR_KR = 2^10 = 0x0000000000000400
- IEEE_2DOT5GBASE_KX = 2^11 = 0x0000000000000800
- IEEE_5GBASE_KR = 2^12 = 0x0000000000001000
- IEEE_50GBASE_CR_KR = 2^13 = 0x0000000000002000
- IEEE_100GBASE_CR2_KR2 = 2^14 = 0x0000000000004000
- IEEE_200GBASE_CR4_KR4 = 2^15 = 0x0000000000008000
- IEEE_100GBASE_CR1_KR1 = 2^16 = 0x0000000000010000
- IEEE_200GBASE_CR2_KR2 = 2^17 = 0x0000000000020000
- IEEE_400GBASE_CR4_KR4 = 2^18 = 0x0000000000040000
- IEEE_800GBASE_CR8_KR8 = 2^19 = 0x0000000000080000
- IEEE_200GBASE_CR1_KR1 = 2^20 = 0x0000000000100000
- IEEE_400GBASE_CR2_KR2 = 2^21 = 0x0000000000200000
- IEEE_800GBASE_CR4_KR4 = 2^22 = 0x0000000000400000
- IEEE_1_6TBASE_CR8_KR8 = 2^23 = 0x0000000000800000
- EC_25GBASE_KR = 2^24 = 0x0000000001000000
- EC_25GBASE_CR = 2^25 = 0x0000000002000000
- EC_50GBASE_KR2 = 2^26 = 0x0000000004000000
- EC_50GBASE_CR2 = 2^27 = 0x0000000008000000
- EC_400GBASE_CR8_KR8 = 2^28 = 0x0000000010000000
- EC_800GBASE_CR8_KR8 = 2^29 = 0x0000000020000000

3. `fec_capable`: *integer*, FEC capability to advertise

- DEFAULT_FEC = 0
- NO_FEC = 1
- FCFEC = 2
- RSFEC_CL91 = 4

- RS528 = 256
- RS544 = 512
- RS272 = 1024

4. fec_requested: *integer*, FEC request to advertise
   - DEFAULT_FEC = 0
   - NO_FEC = 1
   - FCFEC = 2
   - RSFEC_CL91 = 4
   - RS528 = 256
   - RS544 = 512
   - RS272 = 1024
5. pause_mode: *integer*, pause ability to advertise
   - NO_PAUSE = 0
   - SYM_PAUSE = 1
   - ASYM_PAUSE = 2

Example

```
# set
input:  0/1 PP_AUTONEG ANEG_OFF DEFAULT_TECH_MODE DEFAULT_FEC DEFAULT_
↪FEC NO_PAUSE
output: <OK>

# get
input:  0/1 PP_AUTONEG ?
output: 0/1 PP_AUTONEG ANEG_OFF DEFAULT_TECH_MODE DEFAULT_FEC DEFAULT_
↪FEC NO_PAUSE
```

PP_AUTONEGSTATUS

code: 382

Attention: Only for the following modules:

- Thor-400G-7S-1P

```
# get
<module-index>/<port-index> PP_AUTONEGSTATUS ?
```

Description

Status of auto-negotiation settings of the PHY

Actions

get

Parameter

1. `mode:`: *integer*, the mode of autoneg
   - `ANEG_OFF = 0`
   - `ANEG_ON = 1`
2. `fec`: *integer*, the status of auto-negotiation settings of the PHY
   - `PENDING = 0`
   - `NOFEC = 1`
   - `RS_FEC = 513`
   - `FC_FEC = 257`
3. `auto_state`: *integer*, the status of auto-negotiation settings of the PHY
   - `UNKNOWN = 0`
   - `ENABLE = 1`
   - `TRANSMIT_DISABLE = 2`
   - `ABILITY_DETECT = 3`
   - `ACKNOWLEDGE_DETECT = 4`
   - `COMPLETE_ACKNOWLEDGE = 5`
   - `NEXT_PAGE_WAIT = 6`
   - `AN_GOOD_CHECK = 7`
   - `AN_GOOD = 8`
4. `tec_ability`: *integer*, the status of auto-negotiation settings of the PHY
   - `DEFAULT_TECH_MODE = 0`
   - `IEEE_1000BASE_KX = 2^0 = 0x0000000000000001`

- IEEE_10GBASE_KX4 = 2^1 = 0x0000000000000002
- IEEE_10GBASE_KR = 2^2 = 0x0000000000000004
- IEEE_40GBASE_KR4 = 2^3 = 0x0000000000000008
- IEEE_40GBASE_CR4 = 2^4 = 0x0000000000000010
- IEEE_100GBASE_CR10 = 2^5 = 0x0000000000000020
- IEEE_100GBASE_KP4 = 2^6 = 0x0000000000000040
- IEEE_100GBASE_KR4 = 2^7 = 0x0000000000000080
- IEEE_100GBASE_CR4 = 2^8 = 0x0000000000000100
- IEEE_25GBASE_CR_S_KR_S = 2^9 = 0x0000000000000200
- IEEE_25GBASE_CR_KR = 2^10 = 0x0000000000000400
- IEEE_2DOT5GBASE_KX = 2^11 = 0x0000000000000800
- IEEE_5GBASE_KR = 2^12 = 0x0000000000001000
- IEEE_50GBASE_CR_KR = 2^13 = 0x0000000000002000
- IEEE_100GBASE_CR2_KR2 = 2^14 = 0x0000000000004000
- IEEE_200GBASE_CR4_KR4 = 2^15 = 0x0000000000008000
- IEEE_100GBASE_CR1_KR1 = 2^16 = 0x0000000000010000
- IEEE_200GBASE_CR2_KR2 = 2^17 = 0x0000000000020000
- IEEE_400GBASE_CR4_KR4 = 2^18 = 0x0000000000040000
- IEEE_800GBASE_CR8_KR8 = 2^19 = 0x0000000000080000
- IEEE_200GBASE_CR1_KR1 = 2^20 = 0x0000000000100000
- IEEE_400GBASE_CR2_KR2 = 2^21 = 0x0000000000200000
- IEEE_800GBASE_CR4_KR4 = 2^22 = 0x0000000000400000
- IEEE_1_6TBASE_CR8_KR8 = 2^23 = 0x0000000000800000
- EC_25GBASE_KR = 2^24 = 0x0000000001000000
- EC_25GBASE_CR = 2^25 = 0x0000000002000000
- EC_50GBASE_KR2 = 2^26 = 0x0000000004000000
- EC_50GBASE_CR2 = 2^27 = 0x0000000008000000
- EC_400GBASE_CR8_KR8 = 2^28 = 0x0000000010000000
- EC_800GBASE_CR8_KR8 = 2^29 = 0x0000000020000000

5. `fec_capable`: *integer*, the status of auto-negotiation settings of the PHY

- DEFAULT_FEC = 0
- NO_FEC = 1

- FC_FEC = 2
- RSFEC_CL91 = 4
- RS528 = 256
- RS544 = 512
- RS272 = 1024

6. fec_requested: *integer*, the status of auto-negotiation settings of the PHY

- DEFAULT_FEC = 0
- NO_FEC = 1
- FC_FEC = 2
- RSFEC_CL91 = 4
- RS528 = 256
- RS544 = 512
- RS272 = 1024

7. pause_mode: *integer*, the status of auto-negotiation settings of the PHY

- NO_PAUSE = 0
- SYM_PAUSE = 1
- ASYM_PAUSE = 2

Example

```
# get
input:  0/1 PP_AUTONEGSTATUS ?
output: 0/1 PP_AUTONEGSTATUS ANEG_OFF PENDING UNKNOWN DEFAULT_TECH_
 ↪MODE DEFAULT_FEC DEFAULT_FEC NO_PAUSE
```

PP_LINKTRAIN

code: 383

Attention: Only for the following modules:

- Thor-400G-7S-1P

```
# set
<module-index>/<port-index> PP_LINKTRAIN <mode> <pam4_frame_size> <nrz_
 ↪pam4_init_cond> <nrz_preset> <timeout_mode>
```

(continues on next page)

(continued from previous page)

```
# get
<module-index>/<port-index> PP_LINKTRAIN ?
```

Description

Per-port link training settings

Note: This command corresponds to Clause 72. Only `mode` and `timeout_mode` are applicable. The other parameters will be ignored.

Actions

set, get

Parameters

1. `mode`: *byte*, link training mode
   - `AUTO=START_AFTER_AUTONEG = 0`
   - `FORCED_ENABLE=STANDALONE = 1`
   - `DISABLED = 2`
   - `INTERACTIVE = 3`
2. `pam4_frame_size`: *byte*, PAM4 frame size
   - `P16K_FRAME = 0`
   - `P4K_FRAME = 1`
3. `nrz_pam4_init_cond`: *byte* link training init condition
   - `NO_INIT = 0`
   - `INIT_ENABLED = 1`
4. `nrz_preset`: *byte*, NRZ preset
   - `NRZ_NO_PRESET = 0`
   - `NRZ_WITH_PRESET = 1`
5. `timeout_mode`: *byte*, timeout mode
   - `DEFAULT_TIMEOUT = 0`
   - `TIMEOUT_DISABLED = 255`

Example

```
# set
input:  0/1 PP_LINKTRAIN AUTO P16K_FRAME NO_INIT NRZ_NO_PRESET DEFAULT_
 ↪TIMEOUT
output: <OK>

# get
input:  0/1 PP_LINKTRAIN ?
output: 0/1 PP_LINKTRAIN AUTO P16K_FRAME NO_INIT NRZ_NO_PRESET DEFAULT_
 ↪TIMEOUT
```

PP_LINKTRAINSTATUS

code: 384

> **Attention:** Only for the following modules:
>
> - Thor-400G-7S-1P

```
# get
<module-index>/<port-index> PP_LINKTRAINSTATUS [<serdes_index>] ?
```

Description

Per-lane link training status

Actions

get

Parameters

1. `mode`: *byte*, link training status of a lane of a port, including mode, lane status, and failure type.

- `DISABLED = 0`
- `ENABLED = 1`

2. `status`: *byte*, link training status of a lane of a port, including mode, lane status, and failure type.

- `NOT_TRAINED = 0`

- TRAINED = 1

3. failure: *byte*, link training status of a lane of a port, including mode, lane status, and failure type.

- NO_FAILURE = 0
- FRAME_LOCK_ERROR = 1
- SNR_BELOW_THRESHOLD = 2
- TIME_OUT_FAILURE = 3

Example

```
# get
input:  0/1 PP_LINKTRAINSTATUS [0] ?
output: 0/1 PP_LINKTRAINSTATUS [0] DISABLED NOT_TRAINED NO_FAILURE
```

Medium

Status

P_TCVRSTATUS

code: 357

```
# get
<module-index>/<port-index> P_TCVRSTATUS ?
```

Description

Get various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).

Actions

get

Parameters

1. `rx_loss_lane_0`: *short integer*, various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).
2. `rx_loss_lane_1`: *short integer*, various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).
3. `rx_loss_lane_2`: *short integer*, various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).
4. `rx_loss_lane_3`: *short integer*, various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).

Example

```
# get
input:  0/1 P_TCVRSTATUS ?
output: 0/1 P_TCVRSTATUS 123 123 123 123
```

PX_TEMPERATURE

`code:  538`

```
# get
<module-index>/<port-index> PX_TEMPERATURE ?
```

Description

Transceiver temperature in degrees Celsius.

Actions

get

Parameters

1. `integral_part`: *short integer*, temperature value before the decimal digit, and 1/256th of a degree Celsius after the decimal digit.
2. `fractional_part`: *short integer*, temperature value before the decimal digit, and 1/256th of a degree Celsius after the decimal digit.

Example

```
# get
input:  0/1 PX_TEMPERATURE ?
output: 0/1 PX_TEMPERATURE 23 23
```

Transmitter Taps

PL1_PHYTXEQ

code: 442

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_PHYTXEQ [<serdes_index>] <pre3> <pre2>
 ↪<pre> <main> <post>

# get
<module-index>/<port-index> PL1_PHYTXEQ [<serdes_index>] ?
```

Description

Control and monitor the equalizer settings of the on-board PHY in the transmission direction (towards the transceiver cage).

Note: PL1_PHYTXEQ, PL1_PHYTXEQ_LEVEL, and PL1_PHYTXEQ_COEFF facilitate the configuration and retrieval of TX tap values, each offering a unique perspective. Modifications made with any of these parameters will result in updates to the read results across all of them.

Actions

set, get

Parameters

1. `pre3`: *integer*, **pre3** tap value. Default = 0 (neutral)
2. `pre2`: *integer*, **pre2** tap value. Default = 0 (neutral)
3. `pre`: *integer*, **pre** tap value. Default = 0 (neutral)
4. `main`: *integer*, **main** tap value.
5. `post`: *integer*, **post** tap value. Default = 0 (neutral)

Example

```
# set
input:  0/1 PL1_PHYTXEQ [0] 1 4 5 80 3
output: <OK>

# get
input:  0/1 PL1_PHYTXEQ [0] ?
output: 0/1 PL1_PHYTXEQ [0] 1 4 5 80 3
```

PL1_PHYTXEQ_COEFF

`code:  431`

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_PHYTXEQ_COEFF [<serdes_index>] <pre3>
↪<pre2> <pre> <main> <post>

# get
<module-index>/<port-index> PL1_PHYTXEQ_COEFF [<serdes_index>] ?
```

Description

Control and monitor the equalizer settings of taps in the transmission direction (towards the transceiver cage).

Note: `PL1_PHYTXEQ`, `PL1_PHYTXEQ_LEVEL`, and `PL1_PHYTXEQ_COEFF` facilitate the configuration and retrieval of TX tap values, each offering a unique perspective. Modifications made with any of these parameters will result in updates to the read results across all of them.

Actions

set, get

Parameters

1. `pre3`: *integer*, **pre3** tap value, negative, scaled by 1E3. Default = 0 (neutral)
2. `pre2`: *integer*, **pre2** tap value, positive, scaled by 1E3. Default = 0 (neutral)
3. `pre`: *integer*, **pre** tap value, negative, scaled by 1E3. Default = 0 (neutral)
4. `main`: *integer*, **main** tap value, positive, scaled by 1E3. Default = 1
5. `post`: *integer*, **post** tap value, negative, scaled by 1E3. Default = 0 (neutral)

Example

```
# set
input:  0/1 PL1_PHYTXEQ_COEFF [0] -200 100 -100 1000 -100
output: <OK>

# get
input:  0/1 PL1_PHYTXEQ_COEFF [0] ?
output: 0/1 PL1_PHYTXEQ_COEFF [0] -200 100 -100 1000 -100
```

PL1_PHYTXEQ_LEVEL

`code:  430`

Attention: Only for the following modules:

- Freya-800G-4S-1P

- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_PHYTXEQ_LEVEL [<serdes_index>] <pre3>
↪<pre2> <pre> <main> <post>

# get
<module-index>/<port-index> PL1_PHYTXEQ_LEVEL [<serdes_index>] ?
```

Description

Control and monitor the equalizer settings of the on-board PHY in the transmission direction (towards the transceiver cage).

Note: `PL1_PHYTXEQ`, `PL1_PHYTXEQ_LEVEL`, and `PL1_PHYTXEQ_COEFF` facilitate the configuration and retrieval of TX tap values, each offering a unique perspective. Modifications made with any of these parameters will result in updates to the read results across all of them.

Actions

set, get

Parameters

1. `pre3`: *integer*, **pre3** tap value in dB/10, ranges from 0 to 71. Default = 0 (neutral)
2. `pre2`: *integer*, **pre2** tap value in dB/10, ranges from 0 to 71. Default = 0 (neutral)
3. `pre`: *integer*, **pre** tap value in dB/10, ranges from 0 to 187. Default = 0 (neutral)
4. `main`: *integer*, **main** tap value in mV, ranges from 507 to 998.
5. `post`: *integer*, **post** tap value in dB/10, ranges from 0 to 187 Default = 0 (neutral)

Example

```
# set
input:  0/1 PL1_PHYTXEQ_LEVEL [0] 6 9 13 970 15
output: <OK>

# get
```

(continues on next page)

(continued from previous page)

```
input:  0/1 PL1_PHYTXEQ_LEVEL [0] ?
output: 0/1 PL1_PHYTXEQ_LEVEL [0] 6 9 13 970 15
```

PP_PHYAUTOTUNE

code: 360

```
# set
<module-index>/<port-index> PP_PHYAUTOTUNE [<serdes_index>] <on_off>

# get
<module-index>/<port-index> PP_PHYAUTOTUNE [<serdes_index>] ?
```

Description

Enable or disable the automatic receiving of PHY retuning (see PP_PHYRETUNE), which is performed on the 25G interfaces as soon as a signal is detected by the transceiver. Useful if a bad signal causes the PHY to continuously retune or if for some other reason it is preferable to use manual retuning (PP_PHYRETUNE).

Actions

set, get

Parameters

1. on_off: *short integer*, Enable/disable automatic receiving PHY retuning. Default is enabled

Example

```
# set
input:  0/1 PP_PHYAUTOTUNE [0] OFF
output: <OK>

# get
input:  0/1 PP_PHYAUTOTUNE [0] ?
output: 0/1 PP_PHYAUTOTUNE [0] OFF
```

PP_PHYRETUNE

code: 359

```
# set
<module-index>/<port-index> PP_PHYRETUNE [<serdes_index>] <dummy>
```

Description

Trigger a new retuning of the receive equalizer on the PHY for one of the 25G SerDes. Useful if e.g. a direct attached copper cable or loop transceiver does not go into sync after insertion. Note that the retuning will cause disruption of the traffic on all SerDes.

Actions

set

Parameters

1. dummy: *short integer*, reserved for future improvements, always set to 1

Example

```
# set
input:  0/1 PP_PHYRETUNE [0] 1
output: <OK>
```

PP_PHYTXEQ

code: 358

Attention: Only for the following modules:

- Loki-100G-5S-2P
- Thor-400G-7S-1P
- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PP_PHYTXEQ [<serdes_index>] <tap0> <tap1>
↪<tap2> <tap3> <tap4> <tap5> <mode>

# get
<module-index>/<port-index> PP_PHYTXEQ [<serdes_index>] ?
```

Description

Control and monitor the equalizer settings of the on-board PHY in the transmission direction (towards the transceiver cage).

Important: For different module types, the meaning of the tap values are slightly different.

Note: `PP_PHYTXEQ`, `PP_PHYTXEQ_LEVEL`, and `PP_PHYTXEQ_COEFF` facilitate the configuration and retrieval of TX tap values, each offering a unique perspective. Modifications made with any of these parameters will result in updates to the read results across all of them.

Actions

set, get

Parameters

1. `pre`: *integer*, **pre** for Z100 Loki, Z400 Thor, and Z800 Freya. Default = 0 (neutral)
2. `main`: *integer*, **main** for Z100 Loki, Z400 Thor, and Z800 Freya.
3. `post`: *integer*, **post** for Z100 Loki, Z400 Thor, and Z800 Freya. Default = 0 (neutral)
4. `pre2`: *integer*, **N/A** for Z100 Loki, **pre2** for Z400 Thor and Z800 Freya. Default = 0 (neutral)
5. `post2_pre3`: *integer*, **N/A** for Z100 Loki, **post2** for Z400 Thor, **pre3** for Z800 Freya. Default = 0 (neutral)
6. `post3`: *integer*, **N/A** for Z100 Loki, **post3** for Z400 Thor, **N/A** for Z800 Freya. Default = 0 (neutral)
7. `mode`: *integer*, value must be 4

Example

```
# set
input:  0/1 PP_PHYTXEQ [0] 2 80 3 1 4 5 4
output: <OK>

# get
input:  0/1 PP_PHYTXEQ [0] ?
output: 0/1 PP_PHYTXEQ [0] 2 80 3 1 4 5 4
```

Receiver Taps

PP_PHYRXEQ

code: 380

```
# set
<module-index>/<port-index> PP_PHYRXEQ [<serdes_index>] <auto> <ctle>
 ↪<reserved>

# get
<module-index>/<port-index> PP_PHYRXEQ [<serdes_index>] ?
```

Description

RX equalizer parameters.

Actions

set, get

Parameters

1. `auto`: *integer*, auto on or off
2. `ctle`: *integer*, Continuous Time Linear equalization
3. `reserved`: *integer*, reserved

Example

```
# set
input:  0/1 PP_PHYRXEQ [0] 1 1 1
output: <OK>

# get
input:  0/1 PP_PHYRXEQ [0] ?
output: 0/1 PP_PHYRXEQ [0] 1 1 1
```

PP_PHYRXEQ_EXT

code: 397

> **Attention:** Only for the following modules:
>
> - Freya-800G-4S-1P
> - Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PP_PHYRXEQ_EXT [<serdes_index>,
↪<capability_type>] <mode> <value>

# get
<module-index>/<port-index> PP_PHYRXEQ_EXT [<serdes_index>,
↪<capability_type>] ?
```

Description

Get and set RX EQ advanced parameter configurations.

Actions

set, get

Indices

`capability_type`: *integer*, Rx Equalizer Advanced Capability type

- `CTLE_LOW = 0`, Continuous Time Linear Equalization Low
- `CTLE_HIGH = 1`, Continuous Time Linear Equalization High
- `AGC = 2`, Automatic Gain Control
- `OC = 3`, Offset Cancellation
- `CDR = 4`, Clock and Data Recovery
- `PRE_FFE_1 = 5`, Pre Feed-Forward Equalizer #1
- `PRE_FFE_2 = 6`, Pre Feed-Forward Equalizer #2
- `PRE_FFE_3 = 7`, Pre Feed-Forward Equalizer #3
- `PRE_FFE_4 = 8`, Pre Feed-Forward Equalizer #4
- `PRE_FFE_5 = 9`, Pre Feed-Forward Equalizer #5
- `PRE_FFE_6 = 10`, Pre Feed-Forward Equalizer #6
- `PRE_FFE_7 = 11`, Pre Feed-Forward Equalizer #7
- `PRE_FFE_8 = 12`, Pre Feed-Forward Equalizer #8
- `DFE = 13`, Decision Feedback Equalization
- `POST_FFE_1 = 14`, Post Feed-Forward Equalizer #1
- `POST_FFE_2 = 15`, Post Feed-Forward Equalizer #2
- `POST_FFE_3 = 16`, Post Feed-Forward Equalizer #3
- `POST_FFE_4 = 17`, Post Feed-Forward Equalizer #4
- `POST_FFE_5 = 18`, Post Feed-Forward Equalizer #5
- `POST_FFE_6 = 19`, Post Feed-Forward Equalizer #6
- `POST_FFE_7 = 20`, Post Feed-Forward Equalizer #7
- `POST_FFE_8 = 21`, Post Feed-Forward Equalizer #8
- `POST_FFE_9 = 22`, Post Feed-Forward Equalizer #9
- `POST_FFE_10 = 23`, Post Feed-Forward Equalizer #10
- `POST_FFE_11 = 24`, Post Feed-Forward Equalizer #11
- `POST_FFE_12 = 25`, Post Feed-Forward Equalizer #12
- `POST_FFE_13 = 26`, Post Feed-Forward Equalizer #13
- `POST_FFE_14 = 27`, Post Feed-Forward Equalizer #14
- `POST_FFE_15 = 28`, Post Feed-Forward Equalizer #15
- `POST_FFE_16 = 29`, Post Feed-Forward Equalizer #16

- POST_FFE_17 = 30, Post Feed-Forward Equalizer #17
- POST_FFE_18 = 31, Post Feed-Forward Equalizer #18
- POST_FFE_19 = 32, Post Feed-Forward Equalizer #19
- POST_FFE_20 = 33, Post Feed-Forward Equalizer #20
- POST_FFE_21 = 34, Post Feed-Forward Equalizer #21
- POST_FFE_22 = 35, Post Feed-Forward Equalizer #22
- POST_FFE_23 = 36, Post Feed-Forward Equalizer #23

Parameters

1. mode: *integer*, Rx Equalizer Advanced Capability mode
   - AUTO = 0
   - MANUAL = 1
   - FREEZE = 2
2. value: *integer*, the value for the capability

Example

```
# set
input:  0/1 PP_PHYRXEQ_EXT [0,0] MANUAL 31
output: <OK>

# get
input:  0/1 PP_PHYRXEQ_EXT [0,0] ?
output: 0/1 PP_PHYRXEQ_EXT [0,0] MANUAL 31
```

PP_PHYRXEQSTATUS_EXT

code: 398

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# get
<module-index>/<port-index> PP_PHYRXEQSTATUS_EXT [<serdes_index>,
↪<capability_type>] ?
```

Description

Get RX EQ advanced parameter values.

Actions

get

Indices

`capability_type`: *integer*, Rx Equalizer Advanced Capability type

- `CTLE_LOW = 0`, Continuous Time Linear Equalization Low
- `CTLE_HIGH = 1`, Continuous Time Linear Equalization High
- `AGC = 2`, Automatic Gain Control
- `OC = 3`, Offset Cancellation
- `CDR = 4`, Clock and Data Recovery
- `PRE_FFE_1 = 5`, Pre Feed-Forward Equalizer #1
- `PRE_FFE_2 = 6`, Pre Feed-Forward Equalizer #2
- `PRE_FFE_3 = 7`, Pre Feed-Forward Equalizer #3
- `PRE_FFE_4 = 8`, Pre Feed-Forward Equalizer #4
- `PRE_FFE_5 = 9`, Pre Feed-Forward Equalizer #5
- `PRE_FFE_6 = 10`, Pre Feed-Forward Equalizer #6
- `PRE_FFE_7 = 11`, Pre Feed-Forward Equalizer #7
- `DFE = 12`, Decision Feedback Equalization
- `POST_FFE_1 = 13`, Post Feed-Forward Equalizer #1
- `POST_FFE_2 = 14`, Post Feed-Forward Equalizer #2
- `POST_FFE_3 = 15`, Post Feed-Forward Equalizer #3
- `POST_FFE_4 = 16`, Post Feed-Forward Equalizer #4
- `POST_FFE_5 = 17`, Post Feed-Forward Equalizer #5
- `POST_FFE_6 = 18`, Post Feed-Forward Equalizer #6

- `POST_FFE_7 = 19`, Post Feed-Forward Equalizer #7
- `POST_FFE_8 = 20`, Post Feed-Forward Equalizer #8
- `POST_FFE_9 = 21`, Post Feed-Forward Equalizer #9
- `POST_FFE_10 = 22`, Post Feed-Forward Equalizer #10
- `POST_FFE_11 = 23`, Post Feed-Forward Equalizer #11
- `POST_FFE_12 = 24`, Post Feed-Forward Equalizer #12
- `POST_FFE_13 = 25`, Post Feed-Forward Equalizer #13
- `POST_FFE_14 = 26`, Post Feed-Forward Equalizer #14
- `POST_FFE_15 = 27`, Post Feed-Forward Equalizer #15
- `POST_FFE_16 = 28`, Post Feed-Forward Equalizer #16
- `POST_FFE_17 = 29`, Post Feed-Forward Equalizer #17
- `POST_FFE_18 = 30`, Post Feed-Forward Equalizer #18
- `POST_FFE_19 = 31`, Post Feed-Forward Equalizer #19
- `POST_FFE_20 = 32`, Post Feed-Forward Equalizer #20
- `POST_FFE_21 = 33`, Post Feed-Forward Equalizer #21
- `POST_FFE_22 = 34`, Post Feed-Forward Equalizer #22
- `POST_FFE_23 = 35`, Post Feed-Forward Equalizer #23

Parameters

1. `value1`: *integer*, the 1st value for the capability
2. `value2`: *integer*, the 2nd value for the capability

Example

```
# get
input:  0/1 PP_PHYRXEQSTATUS_EXT [0,0] ?
output: 0/1 PP_PHYRXEQSTATUS_EXT [0,0] 16 15
```

Signal Integrity

PL1_CTRL

code: 424

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CTRL [<serdes_index>, <func>] <opcode>
```

Description

The Signal Integrity feature offers the equivalent of an Equivalent Time oscilloscope trace of the RX *PAM4* signal (later, also PAM2). The trace is done with the A/D converter in the GTM receiver also doing the data sampling / CDR function, i.e. the trace is taken after the RX equalizer.

The HW characteristics of the Versal GTM used in Z800 Freya are: Trace length = 2000 samples, sample resolution = 7 bits 2's complement, i.e. range = -64..63.

Using the sampled eye scan feature through CLI involves two steps:

- Trigger the acquisition of a trace (`PL1_CTRL`)
- Retrieve the trace data (`PL1_GET_DATA`)

This command is a generic control function related to Layer 1 / SERDES. For now, only used for signal integrity scan.

Actions

set

Indices

func: *integer*, function to operate on.

- SAMPLED_SIGNAL_INTEGRITY_SCAN = 0

Parameters

1. opcode: *byte*, operation to perform.
    - START_SCAN = 0 for sampled eye scan

Example

```
# set (Initiate a trace scan on port 0/0 SERDES 3)
input:  0/0 PL1_CTRL [3,0] 0
output: <OK>
```

PL1_GET_DATA

code: 425

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PL1_CTRL [<serdes_index>, <func>] <result>
↪<sweepno> <age_us> <value>
```

Description

The Signal Integrity feature offers the equivalent of an Equivalent Time oscilloscope trace of the RX PAM4 signal (later, also PAM2). The trace is done with the A/D converter in the GTM receiver also doing the data sampling / CDR function, i.e. the trace is taken after the RX equalizer.

The HW characteristics of the Versal GTM used in Z800 Freya are: Trace length = 2000 samples, sample resolution = 7 bits 2's complement, i.e. range = -64..63.

Using the sampled eye scan feature through CLI involves two steps:

- Trigger the acquisition of a trace (PL1_CTRL)

- Retrieve the trace data (`PL1_GET_DATA`)

This command is a generic function to retrieve dynamic data related to Layer 1 / SERDES. For now, only used for signal integrity scan.

Actions

set

Indices

`func`: *integer*, function to operate on.

- `SAMPLED_SIGNAL_INTEGRITY_SCAN = 0`

Parameters

1. `result`: *integer*, data availability.
2. `sweep_no`: *integer*, per-SERDES trace acquisition counter.
3. `age_us`: *integer*, the "age" of the trace data in microseconds, i.e. the time from data acquisition from hardware was completed until the time the command reply data is generated.
4. `value`: *byte list*, a set of 16 bit signed 2-complement sample values. With present hardware, the range of each sample is -64..63. In CLI scripting, each sample value is represented as two bytes, msb first.

For `func==0`, sampled eye scan:

- `result==0`: No data available.

"No data available" means that either a scan was never started, an acquisition was started and in progress, or the acquired data has become too old (e.g. older than 500 ms). The acquisition time for a trace is in the very low ms-range. If `result==0`, `sweep_no` and `age_us` are dummy (=`0`), and no additional data are returned.

- `result==1`: Data returned. In that case, the rest of the parameters apply:

`sweep_no`: per-SERDES trace acquisition counter: 1,2,3… Each trace can be returned multiple times, to different users, within its lifetime. A new trace acquisition is triggered with the `PL1_CTRL` command.

`age_us`: The "age" of the trace data in microseconds, i.e. the time from data acquisition from hardware was completed until the time the command reply data is generated.

`value`: The rest of the reply is a set of 16 bit signed 2-complement sample values. With present hardware, the range of each sample is -64..63. In XMP scripting, each sample value is represented as two bytes, msb first.

With present implementation, 2006 sample values (4012 bytes) are returned.

The first 6 sample values are so-called "sampled levels": <p1> <p2> < p3> <m1> <m2> <m3>

Their meaning is explained by the following figure, where the "sampled levels" are illustrated by horizontal lines:

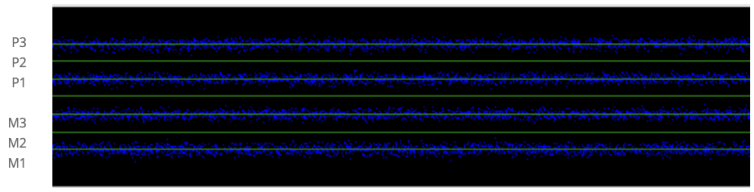


Fig. 3.15: Illustration of sampled levels

M2 and P2 are "discrimination levels". The 3rd discrimination level, the line at 0-level, is fixed and not acquired as part of the trace data.

P1, P3, M1, M3 are the average of each *PAM4* level.

For *PAM2/NRZ*, only P1 and M1 are valid.

The remaining 2000 sample values in the reply are the trace itself - sampled values of the RX signal.

Example

```
# get
input:  0/0 PL1_GET_DATA [3,0] ?
```

PP_EYEBER

code: 361

```
# get
<module-index>/<port-index> PP_EYEBER [<serdes_index>] ?
```

Description

Obtain BER estimations of an eye diagram.

Actions

get

Parameters

1. `eye_ber_estimation`: *string*, BER estimations of an eye diagram

Example

```
# get
input:  0/1 PP_EYEBER [0] ?
output: 0/1 PP_EYEBER [0] "N/A N/A N/A N/A "
```

PP_EYEDWELLBITS

`code:  367`

```
# set
<module-index>/<port-index> PP_EYEDWELLBITS [<serdes_index>] <min_
↪dwell_bit_count> <max_dwell_bit_count>

# get
<module-index>/<port-index> PP_EYEDWELLBITS [<serdes_index>] ?
```

Description

Min and max dwell bits for an eye capture.

Actions

set, get

Parameters

1. `min_dwell_bit_count`: *integer*, minimum dwell bits for an eye capture
2. `max_dwell_bit_count`: *integer*, maximum dwell bits for an eye capture

Example

```
# set
input:  0/1 PP_EYEDWELLBITS [0] 1 1
output: <OK>

# get
input:  0/1 PP_EYEDWELLBITS [0] ?
output: 0/1 PP_EYEDWELLBITS [0] 1 1
```

PP_EYEINFO

`code:  356`

```
# get
<module-index>/<port-index> PP_EYEINFO [<serdes_index>] ?
```

Description

Read out BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes. This must be called after "PP_EYEMEASURE" has run to return valid results. Use "get" to see the status of the data gathering process.

Actions

get

Parameters

1. `width_mui`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
2. `height_mv`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
3. `h_slope_left`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes

4. `h_slope_right`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
5. `y_intercept_left`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
6. `y_intercept_right`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
7. `r_squared_fit_left`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
8. `r_squared_fit_right`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
9. `est_rj_rms_left`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
10. `est_rj_rms_right`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
11. `est_dj_pp`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
12. `v_slope_bottom`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
13. `v_slope_top`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
14. `x_intercept_bottom`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
15. `x_intercept_top`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
16. `r_squared_fit_bottom`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
17. `r_squared_fit_top`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
18. `est_rj_rms_bottom`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes
19. `est_rj_rms_top`: *integer*, BER eye-measurement information such as the vertical and horizontal bathtub curve information on a 25G SerDes

Example

```
# get
input:  0/1 PP_EYEINFO [0] ?
output: 0/1 PP_EYEINFO [0] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

PP_EYEMEASURE

code: 353

```
# set
<module-index>/<port-index> PP_EYEMEASURE [<serdes_index>] <status>
↪<dummy>

# get
<module-index>/<port-index> PP_EYEMEASURE [<serdes_index>] ?
```

Description

Start/stop a new BER eye-measure on a 25G SerDes. Use "get" to see the status of the data gathering process.

Actions

set, get

Parameters

1. `status`: *byte*, action (when set)
   - `STOP = 0`
   - `START = 1`
2. `status`: *byte*, status of the SerDes (when get)
   - `STOPPED = 0`
   - `STARTED = 1`
   - `INITIALIZING = 2`
   - `PLOTTING = 3`
3. `dummy`: *short integer list*, should always be 0, reserved for future expansion

Example

```
# set
input:  0/1 PP_EYEMEASURE [0] STOP 0
output: <OK>

# get
input:  0/1 PP_EYEMEASURE [0] ?
output: 0/1 PP_EYEMEASURE [0] STOP 0
```

PP_EYEREAD

code: 355

```
# get
<module-index>/<port-index> PP_EYEREAD [<serdes_index>, <column_index>
↪] ?
```

Description

Read a single column of a measured BER eye on a 25G SerDes. Every readout also returns the resolution (x,y) and the number of valid columns (used to facilitate reading out the eye while it is being measured). Note that the columns of the eye-data will be measured in the order: xres-1, xres-2, xres-3, … 0. The values show the number of bit errors measured out of a total of 1M bits at each of the individual sampling points (x=timeaxis, y = 0/1 threshold).

Actions

get

Parameters

1. `x_resolution`: *integer*, x resolution, y resolution, number of valid columns, and the number of bit errors measured out of a total of 1M bits at each of the individual sampling points (x=timeaxis, y = 0/1 threshold).
2. `y_resolution`: *integer*, x resolution, y resolution, number of valid columns, and the number of bit errors measured out of a total of 1M bits at each of the individual sampling points (x=timeaxis, y = 0/1 threshold).
3. `valid_column_count`: *integer*, x resolution, y resolution, number of valid columns, and the number of bit errors measured out of a total of 1M bits at each of the individual sampling points (x=timeaxis, y = 0/1 threshold).

4. `values`: *integer list*, x resolution, y resolution, number of valid columns, and the number of bit errors measured out of a total of 1M bits at each of the individual sampling points (x=timeaxis, y = 0/1 threshold).

Example

```
# get
input:  0/1 PP_EYEREAD [0, 0] ?
output: 0/1 PP_EYEREAD [0, 0] 1 1 1 0 1
```

PP_EYERESOLUTION

`code:  354`

```
# set
<module-index>/<port-index> PP_EYERESOLUTION [<serdes_index>] <x_
↪resolution> <y_resolution>

# get
<module-index>/<port-index> PP_EYERESOLUTION [<serdes_index>] ?
```

Description

Set or get the resolution used for the next BER eye-measurement.

Actions

set, get

Parameters

1. *x_resolution* `: *integer*, number of columns, must be between 9 and 65 and be in the form 2^n+1
2. `y_resolution`: *integer*, number of columns, must be between 7 and 255 and be in the form 2^n-1

Example

```
# set
input:  0/1 PP_EYERESOLUTION [0] 1 1
output: <OK>

# get
input:  0/1 PP_EYERESOLUTION [0] ?
output: 0/1 PP_EYERESOLUTION [0] 1 1
```

PP_PHYSIGNALSTATUS

code: 375

```
# get
<module-index>/<port-index> PP_PHYSIGNALSTATUS ?
```

Description

Obtain the PHY signal status.

Actions

get

Parameters

1. phy_signal_status: *byte*, PHY signal status

- NO_SIGNAL = 0
- NO_CDRLOCK = 2
- LOCKED = 3

Example

```
# get
input:  0/1 PP_PHYSIGNALSTATUS ?
output: 0/1 PP_PHYSIGNALSTATUS NO_SIGNAL
```

Transceiver Register

PX_I2C_CONFIG

code: 539

Attention: Only for the following modules:

- Freya-800G-4S-1P
- Freya-800G-4S-1P-OSFP

```
# set
<module-index>/<port-index> PX_I2C_CONFIG <frequency>

# get
<module-index>/<port-index> PX_I2C_CONFIG ?
```

Description

Set the get the access speed on a transceiver *I2C* access in the unit of KHz. Default to 100. When the transceiver is plugged out and in again, the speed will be reset to the default value 100. The speed has a minimum and a maximum, which can be obtained from `P_CAPABILITIES`. The I2C speed configuration will not be included in the port configuration file (.xpc). When you load a port configuration to a port, the transceiver I2C access speed will be reset to default 100.

Actions

set, get

Parameters

1. `frequency`: *integer*, the speed on a transceiver I2C access in the unit of KHz. Default to 100. When the transceiver is plugged out and in again, this parameter will be reset to the default value 100. This parameter has minimum and maximum limits, which can be obtained from `P_CAPABILITIES`.

Example

```
# set
input: 0/1 PX_I2C_CONFIG 1000
output: <OK>

# get
input:  0/1 PX_I2C_CONFIG ?
output: 0/1 PX_I2C_CONFIG 1000
```

PX_MII

code: 537

```
# set
<module-index>/<port-index> PX_MII [<address>] <value>

# get
<module-index>/<port-index> PX_MII [<address>] ?
```

Description

Provides access to the register interface supported by the media-independent interface (MII) transceiver. It is possible to both read and write register values. <address>: the transceiver address, integer, 0x00 - 0xFF (0-255).

Actions

set, get

Parameters

1. value: *hex2*, register value of a transceiver

Example

```
# set
input:  0/1 PX_MII [0x00] 0x0011
output: <OK>

input:  0/1 PX_MII [0] 0x0011
output: <OK>

# get
input:  0/1 PX_MII [0x00] ?
output: 0/1 PX_MII [0x00] 0x0011

input:  0/1 PX_MII [0] ?
output: 0/1 PX_MII [0] 0x0011
```

PX_RW

code: 501

```
# set
<module-index>/<port-index> PX_RW [<page>, <address>] <value>

# get
<module-index>/<port-index> PX_RW [<page>, <address>] ?
```

Description

Provides access to the register interface supported by the port transceiver. It is possible to both read and write register values. <page>: the transceiver page address, integer, 0x00 - 0xFF (0-255). <address>: the address within the page, integer, 0x00 - 0xFF (0-255).

Actions

set, get

Parameters

1. `value`: *hex4*, register value of a transceiver.

Example

```
# set
input:  0/1 PX_RW [0x00,0x56] 0x0000000F
output: <OK>

input:  0/1 PX_RW [0,86] 0x0000000F
output: <OK>

# get
input:  0/1 PX_RW [0x00,0x56] ?
output: 0/1 PX_RW [0x00,0x56] 0x00000011

input:  0/1 PX_RW [0,86] ?
output: 0/1 PX_RW [0,86] 0x00000011
```

PX_RW_SEQ

`code:  503`

```
# set
<module-index>/<port-index> PX_RW_SEQ [<page>, <address>, <byte_count>
↪] <value>

# get
<module-index>/<port-index> PX_RW_SEQ [<page>, <address>, <byte_count>
↪] ?
```

Description

I2C sequential access to a transceiver's register. When invoked, the <byte_count> number of bytes will be read or written in one I2C transaction, in which the <value> is read or written with only a single register address setup. A subsequent invocation will perform a second I2C transaction in the same manner. <page>: the transceiver page address, integer, 0x00 - 0xFF (0-255). <address>: the address within the page, integer, 0x00 - 0xFF (0-255).

Actions

set, get

Parameters

1. value: *hex list*, the bytes to be read or written in one I2C transaction. The number of bytes in the <value> equals <byte_count>.

Example

```
# set
input: 0/1 PX_RW_SEQ [0x00,0x14,0x0A] 0x00001111222200001111
output: <OK>

input: 0/1 PX_RW_SEQ [0,20,10] 0x00001111222200001111
output: <OK>

# get
input:  0/1 PX_RW_SEQ [0x00,0x14,0x0A] ?
output: 0/1 PX_RW_SEQ [0x00,0x14,0x0A] 0x00001111222200001111

input:  0/1 PX_RW_SEQ [0,20,10] ?
output: 0/1 PX_RW_SEQ [0,20,10] 0x00001111222200001111
```

PX_RW_SEQ_EXT

code: 504

```
# set
<module-index>/<port-index> PX_RW_SEQ [<bank>, <page>, <address>,
↪<byte_count>] <value>

# get
```

(continues on next page)

(continued from previous page)

```
<module-index>/<port-index> PX_RW_SEQ [<bank>, <page>, <address>,
↪<byte_count>] ?
```

Description

I2C sequential access to a transceiver's register. When invoked, the <byte_count> number of bytes will be read or written in one I2C transaction, in which the <value> is read or written with only a single register address setup. A subsequent invocation will perform a second I2C transaction in the same manner. <bank>: the bank address, integer, 0x00 - 0xFF (0-255). <page>: the transceiver page address, integer, 0x00 - 0xFF (0-255). <address>: the address within the page, integer, 0x00 - 0xFF (0-255).

Actions

set, get

Parameters

1. value: *hex list*, the bytes to be read or written in one I2C transaction. The number of bytes in the <value> equals <byte_count>.

Example

```
# set
input: 0/1 PX_RW_SEQ [0x00,0x01,0x14,0x0A] 0x00001111222200001111
output: <OK>

input: 0/1 PX_RW_SEQ [0,1,20,10] 0x00001111222200001111
output: <OK>

# get
input:  0/1 PX_RW_SEQ [0x00,0x01,0x14,0x0A] ?
output: 0/1 PX_RW_SEQ [0x00,0x01,0x14,0x0A] 0x00001111222200001111

input:  0/1 PX_RW_SEQ [0,1,20,10] ?
output: 0/1 PX_RW_SEQ [0,1,20,10] 0x00001111222200001111
```

Laser Power

PP_RXLASERPOWER

code: 295

```
# get
<module-index>/<port-index> PP_RXLASERPOWER ?
```

Description

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 type of CFP transceiver used. The list is empty if the CFP transceiver does not support optical power read-out.

Actions

get

Parameters

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

Example

```
# get
input:  0/1 PP_RXLASERPOWER ?
output: 0/1 PP_RXLASERPOWER 0 1
```

PP_TXLASERPOWER

code: 296

```
# get
<module-index>/<port-index> PP_TXLASERPOWER ?
```

Description

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

Actions

get

Parameters

1. `nanowatts`: *integer list*, received signal level, in nanowatts. 0, when no signal.

Example

```
# get
input:  0/1 PP_TXLASERPOWER ?
output: 0/1 PP_TXLASERPOWER 0 1
```

BroadR-Reach

P_BRRMODE

`code:  326`

```
# set
<module-index>/<port-index> P_BRRMODE <mode>

# get
<module-index>/<port-index> P_BRRMODE ?
```

Description

Selects the Master/Slave setting of 100 Mbit/s (requires Valkyrie release 76.1 or higher) and 1000 Mbit/s (requires Valkyrie release 76.2 or higher) BroadR-Reach copper interfaces.

Actions

set, get

Parameters

1. mode: *byte*, the port's BroadR-Reach mode

- SLAVE = 0
- MASTER = 1

Example

```
# set
input:  0/1 P_BRRMODE SLAVE
output: <OK>

# get
input:  0/1 P_BRRMODE ?
output: 0/1 P_BRRMODE SLAVE
```

MDI/MDIX

P_MDIXMODE

code: 194

```
# set
<module-index>/<port-index> P_MDIXMODE <mode>

# get
<module-index>/<port-index> P_MDIXMODE ?
```

Description

Selects the MDI/MDIX behavior of copper interfaces (Currently supported on M6SFP and M2SFPT).

Actions

set, get

Parameters

1. mode: *byte*, the MDI/MDIX mode of the port.

- AUTO = 0
- MDI = 1
- MDIX = 2

Example

```
# set
input:  0/1 P_MDIXMODE AUTO
output: <OK>

# get
input:  0/1 P_MDIXMODE ?
output: 0/1 P_MDIXMODE AUTO
```

Config

PP_CONFIG

code: 297

```
# get
<module-index>/<port-index> PP_CONFIG ?
```

Description

Return all setable L1 values for port

Actions

get

Parameters

Example

```
# get
input:  0/1 PP_CONFIG ?
```

3.3.4 Impairment

Flows

Flows in E100 Chimera are identified using a *flow ID* (fid). Valid fids are 0-7, where fid = 0 is the port default flow.

When configuring the flow filters and flow impairments, the fid of the flow to be configured must be provided to the API using square brackets after the API command. E.g: `PEF_INIT [fid]`.

Filters

A filter configuration can either be Basic (the default), or Extended. Basic mode enables configuration of a basic subset of protocol segments in a fixed order whereas Extended allows a much high degree of flexibility to filter within the first 128 bytes of a packet. The mode is selected with *PEF_MODE*. Basic mode allows filtering on:

- Ethernet header
- Up to two *VLANs*
- Up to one *MPLS* label
- The first IPv4 or IPv6 header
- Up to six sequential bytes of payload anywhere within the first 128 bytes of the packet
- Xena Test Payload IDs (TPID)

Extended mode allows filtering based on an initial Ethernet header followed by any combination of segments up to a total length of 128 bytes, as well as optionally Xena Test Payload IDs (TPID).

For example:

0/0 PEF_PROTOCOL [1,0] ETHERNET VLAN VLAN MPLS MPLS -4 ETHERNET VLAN`` could specify a Metro Ethernet Forum provider-tagged (first VLAN) customer-tagged (second

VLAN) frame containing an MPLS tunnel (first MPLS label) containing an MPLS Pseudo Wire (second MPLS label + four bytes of Control Word) containing a single-tagged Ethernet frame.

Once the protocol layout has been specified it is possible to set up a sequence of value + mask bytes that selects the relevant fields of each protocol segment; see *PEF_VALUE* and *PEF_MASK*.

CLI for Impairment

CLI commands for impairment (E100 Chimera) functionalities:

Impairment Configuration

P_EMULATE

`code: 1600`

```
# set
<module-index>/<port-index> P_EMULATE <action>

# get
<module-index>/<port-index> P_EMULATE ?
```

Description

The action determines if emulation functionality is enabled or disabled

Actions

set, get

Parameters

1. `action`: *byte*, whether the E100 Chimera port's emulation functionality is enabled
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 P_EMULATE OFF
output: <OK>

# get
input:  0/1 P_EMULATE ?
output: 0/1 P_EMULATE OFF
```

P_LOADMODE

code: 395

```
# set
<module-index>/<port-index> P_LOADMODE <on_off>

# get
<module-index>/<port-index> P_LOADMODE ?
```

Description

The action determines if config load mode is enabled or disabled for the E100 Chimera port.

Actions

set, get

Parameters

1. `on_off`: *byte*, whether config load is enabled on the E100 Chimera port

- `OFF = 0`
- `ON = 1`

Example

```
# set
input:  0/1 P_LOADMODE OFF
output: <OK>

# get
input:  0/1 P_LOADMODE ?
output: 0/1 P_LOADMODE OFF
```

PE_BANDPOLICER

code: 1662

```
# set
<module-index>/<port-index> PE_BANDPOLICER [<flow_index>] <on_off>
↪<mode> <cir> <cbs>

# get
<module-index>/<port-index> PE_BANDPOLICER [<flow_index>] ?
```

Description

Configures the bandwidth policer.

Actions

set, get

Parameters

1. `on_off`: *byte*, enables/disables policer. Note: PED_ENABLE is not supported for the policer.

- `OFF = 0`
- `ON = 1`

2. `mode`: *byte*, policer mode.

- `L1 = 0`
- `L2 = 1`

3. `cir`: *integer*, policer committed information rate in units of 100 kbps (range 0 to 1000000), default is 0.

4. `cbs`: *integer*, policer committed burst burst in bytes (range 0 to 4194304), default is 0.

Example

```
# set
input:  0/1 PE_BANDPOLICER [0] OFF L1 1 1
output: <OK>

# get
input:  0/1 PE_BANDPOLICER [0] ?
output: 0/1 PE_BANDPOLICER [0] OFF L1 1 1
```

PE_BANDSHAPER

`code:  1663`

```
# set
<module-index>/<port-index> PE_BANDSHAPER [<flow_index>] <on_off>
 ↪<mode> <cir> <cbs> <buffer_size>

# get
<module-index>/<port-index> PE_BANDSHAPER [<flow_index>] ?
```

Description

Configures the bandwidth shaper. L1 (0) (Shaper performed at Layer 1 level. I.e. including the preamble and min interpacket gap) L2 (1) (Shaper performed at Layer 2 level. I.e. excluding the preamble and min interpacket gap) Default value: L2 (0)

Actions

set, get

Parameters

1. `on_off`: *byte*, enables/disables shaper.
   - `OFF = 0`
   - `ON = 1`
2. `mode`: *byte*, shaper mode.
   - `L1 = 0`

- L2 = 1

3. cir: *integer*, shaper committed information rate in units of 100 kbps (range 0 to 1000000), default is 0.
4. cbs: *integer*, shaper committed burst size in bytes (range 0 to 4194304), default is 0.
5. buffer_size: *integer*, shaper buffer size in bytes (range 0 to 2097152), default is 0.

Example

```
# set
input:  0/1 PE_BANDSHAPER [0] OFF L1 1 1 1
output: <OK>

# get
input:  0/1 PE_BANDSHAPER [0] ?
output: 0/1 PE_BANDSHAPER [0] OFF L1 1 1 1
```

PE_COMMENT

code: 1605

```
# set
<module-index>/<port-index> PE_COMMENT [<flow_index>] <comment>

# get
<module-index>/<port-index> PE_COMMENT [<flow_index>] ?
```

Description

Flow description.

Actions

set, get

Parameters

1. comment: *string*, the description of the flow

Example

```
# set
input:  0/1 PE_COMMENT [0] 'A String'
output: <OK>

# get
input:  0/1 PE_COMMENT [0] ?
output: 0/1 PE_COMMENT [0] 'A String'
```

PE_CONFIG

code: 1606

```
# get
<module-index>/<port-index> PE_CONFIG [<flow_index>] ?
```

Description

Return flow config

Actions

get

Parameters

Example

```
# get
input:  0/1 PE_CONFIG [0] ?
```

PE_CORRUPT

code: 1660

```
# set
<module-index>/<port-index> PE_CORRUPT [<flow_index>] <corruption_type>

# get
<module-index>/<port-index> PE_CORRUPT [<flow_index>] ?
```

Description

Configures impairment corruption type.

Note: IP / TCP / UDP corruption modes are not supported on default flow (0)

Actions

set, get

Parameters

1. corruption_type: *byte*, corruption type

- OFF = 0
- ETH = 1
- IP = 2
- UDP = 3
- TCP = 4
- BER = 5

Example

```
# set
input:  0/1 PE_CORRUPT [0] OFF
output: <OK>

# get
input:  0/1 PE_CORRUPT [0] ?
output: 0/1 PE_CORRUPT [0] OFF
```

PE_FCSDROP

code: 1601

```
# set
<module-index>/<port-index> PE_FCSDROP <on_off>

# get
<module-index>/<port-index> PE_FCSDROP ?
```

Description

The action on packets with FCS errors on a port.

Actions

set, get

Parameters

1. on_off: *byte*, whether the action on packets with FCS errors on a port is enabled
   - OFF = 0
   - ON = 1

Example

```
# set
input:  0/1 PE_FCSDROP OFF
output: <OK>

# get
input:  0/1 PE_FCSDROP ?
output: 0/1 PE_FCSDROP OFF
```

PE_FULLCONFIG

code: 1607

```
# get
<module-index>/<port-index> PE_FULLCONFIG ?
```

Description

Get flow config for all the flows of a port

Actions

get

Parameters

Example

```
# get
input: 0/1 PE_FULLCONFIG ?
```

PE_INDICES

code: 1608

```
# get
<module-index>/<port-index> PE_INDICES ?
```

Description

Get the flow indices. Currently the number of flows is 8.

Actions

get

Parameters

1. `flow0_id`: *integer*, the flow indices of a port
2. `flow1_id`: *integer*, the flow indices of a port
3. `flow2_id`: *integer*, the flow indices of a port
4. `flow3_id`: *integer*, the flow indices of a port
5. `flow4_id`: *integer*, the flow indices of a port
6. `flow5_id`: *integer*, the flow indices of a port
7. `flow6_id`: *integer*, the flow indices of a port
8. `flow7_id`: *integer*, the flow indices of a port

Example

```
# get
input:  0/1 PE_INDICES ?
output: 0/1 PE_INDICES 1 1 1 1 1 1 1 1
```

PE_LATENCYRANGE

code: 1646

```
# get
<module-index>/<port-index> PE_LATENCYRANGE [<flow_index>] ?
```

Description

Retrieve minimum and maximum configurable latency per flow in nanoseconds.

Actions

get

Parameters

1. `min_latency`: *long integer*, minimum and maximum configurable latency per flow in nanoseconds.
2. `max_latency`: *long integer*, minimum and maximum configurable latency per flow in nanoseconds.

Example

```
# get
input:  0/1 PE_LATENCYRANGE [0] ?
output: 0/1 PE_LATENCYRANGE [0] 123456789123 123456789123
```

PE_MISORDER

`code:  1661`

```
# set
<module-index>/<port-index> PE_MISORDER [<flow_index>] <depth>

# get
<module-index>/<port-index> PE_MISORDER [<flow_index>] ?
```

Description

Configures the misordering depth in number of packets.

Note: probability [see *PED_FIXED*] * (depth + 1) should be less than 1,000,000.

Actions

set, get

Parameters

1. `depth`: *integer*, the misordering depth (Range 1 - 32). Default value.

Example

```
# set
input:  0/1 PE_MISORDER [0] 1
output: <OK>

# get
input:  0/1 PE_MISORDER [0] ?
output: 0/1 PE_MISORDER [0] 1
```

PE_TPLDMODE

`code:  1602`

```
# set
<module-index>/<port-index> PE_TPLDMODE <mode>

# get
<module-index>/<port-index> PE_TPLDMODE ?
```

Description

The action indicates the TPLD mode to be used per port.

Actions

set, get

Parameters

1. mode: *byte*, indicating the TPLD mode
   - NORMAL = 0
   - MICRO = 1

Example

```
# set
input:  0/1 PE_TPLDMODE NORMAL
output: <OK>

# get
input:  0/1 PE_TPLDMODE ?
output: 0/1 PE_TPLDMODE NORMAL
```

Default Distribution

PED_ACCBURST

code: 1641

```
# set
<module-index>/<port-index> PED_ACCBURST [<flow_index>, <impairment_
↪type_index>] <delay>

# get
<module-index>/<port-index> PED_ACCBURST [<flow_index>, <impairment_
↪type_index>] ?
```

Description

Configuration of Accumulate & Burst distribution (DELAY only).

Note: If the delay is less than minimum latency, value is set to minimum latency. If the delay is greater than maximum latency, value is set to maximum latency.

Actions

set, get

Parameters

1. `delay`: *long integer*, specifies the burst delay time. Units = nanosecond (must multiples of 100 ns). Default value: minimum latency.

Example

```
# set
input:  0/1 PED_ACCBURST [0, 0] 1
output: <OK>

# get
input:  0/1 PED_ACCBURST [0, 0] ?
output: 0/1 PED_ACCBURST [0, 0] 1
```

PED_BER

code: 1623

```
# set
<module-index>/<port-index> PED_BER [<flow_index>, <impairment_type_
↪index>] <coef> <exp>

# get
<module-index>/<port-index> PED_BER [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Bit Error Rate distribution.

Actions

set, get

Parameters

1. `coef`: *integer*, specifies the coefficient for BER. Default value: 1 (Range is 1 to 9).
2. `exp`: *integer*, specifies the exponent for BER. Default value: -10 (Range is -18 to -1).

Example

```
# set
input:  0/1 PED_BER [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_BER [0, 0] ?
output: 0/1 PED_BER [0, 0] 1 1
```

PED_CONST

`code:  1640`

```
# set
<module-index>/<port-index> PED_CONST [<flow_index>, <impairment_type_
↪index>] <delay>

# get
<module-index>/<port-index> PED_CONST [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Constant Delay distribution (DELAY only). Unit is ns (must be multiples of 100ns). Default value: Minimum supported per speed and FEC mode.

Note: If the latency is less than minimum latency, value is set to minimum latency. If the latency is greater than maximum latency, value is set to maximum latency.

Actions

set, get

Parameters

1. `delay`: *long integer*, specifies the constant delay/latency time. Unit is nanosecond (must be multiples of 100 ns). Default value: Minimum supported per speed and FEC mode.

Example

```
# set
input:  0/1 PED_CONST [0, 0] 1
output: <OK>

# get
input:  0/1 PED_CONST [0, 0] ?
output: 0/1 PED_CONST [0, 0] 1
```

PED_CUST

`code:  1631`

```
# set
<module-index>/<port-index> PED_CUST [<flow_index>, <impairment_type_
↪index>] <cust_id>

# get
<module-index>/<port-index> PED_CUST [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Associate a custom distribution to a flow and impairment type.

Note: Before associating a custom distribution, the below validation checks are applied.

In case of `impairment_type_index != DELAY`, (1) Custom values should be less than or equal to max allowed (4194288). (2) Custom distribution bust contain 512 values.

In case of `impairment_type_index = DELAY`, (1) Custom values should be less than or equal to the maximum latency. (2) Custom values should be greater than or equal to minimum latency. (3) Custom distribution should contain 1024 values.

Actions

set, get

Parameters

1. `cust_id`: *integer*, custom distribution identifier

Example

```
# set
input:  0/1 PED_CUST [0, 0] 1
output: <OK>

# get
input:  0/1 PED_CUST [0, 0] ?
output: 0/1 PED_CUST [0, 0] 1
```

PED_ENABLE

`code:  1644`

```
# get
<module-index>/<port-index> PED_ENABLE [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Control whether impairment is enabled of disabled.

Note: This command is not applicable for *PE_BANDPOLICER* and *PE_BANDSHAPER* because they have a separate `ON / OFF` parameter.

Actions

get

Parameters

1. `action`: *short integer*, whether impairment is enabled of disabled

Example

```
# get
input:  0/1 PED_ENABLE [0, 0] ?
output: 0/1 PED_ENABLE [0, 0] 123
```

PED_FIXED

`code:  1621`

```
# set
<module-index>/<port-index> PED_FIXED [<flow_index>, <impairment_type_
↪index>] <probability>

# get
<module-index>/<port-index> PED_FIXED [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Fixed Rate distribution. This is predictable distribution with nearly equal distance between impairments, to match the configured probability.

Note: In case of misordering, a special limit applies, probability * (depth + 1) should be less than 1000000.

Actions

set, get

Parameters

1. `probability`: *integer*, the fixed probability in ppm. Default value is 0.

Example

```
# set
input:  0/1 PED_FIXED [0, 0] 1
output: <OK>

# get
input:  0/1 PED_FIXED [0, 0] ?
output: 0/1 PED_FIXED [0, 0] 1
```

PED_FIXEDBURST

`code:  1624`

```
# set
<module-index>/<port-index> PED_FIXEDBURST [<flow_index>, <impairment_
↪type_index>] <burst_size>

# get
<module-index>/<port-index> PED_FIXEDBURST [<flow_index>, <impairment_
↪type_index>] ?
```

Description

Configuration of Fixed Burst distribution.

Note: In case of `impairment_type_index` = `MISO`, burstsize is fixed to 1.

Actions

set, get

Parameters

1. `burst_size`: *integer*, specifies the burst size (Range 1 - 16383). Default value = 1.

Example

```
# set
input:  0/1 PED_FIXEDBURST [0, 0] 1
output: <OK>

# get
input:  0/1 PED_FIXEDBURST [0, 0] ?
output: 0/1 PED_FIXEDBURST [0, 0] 1
```

PED_GAMMA

`code:  1630`

```
# set
<module-index>/<port-index> PED_GAMMA [<flow_index>, <impairment_type_
↪index>] <shape> <scale>

# get
<module-index>/<port-index> PED_GAMMA [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Gamma distribution.

Note: Mean and Standard deviation are calculated from Shape and Scale parameters and validation is performed using those. standard deviation = [SQRT(shape * scale * scale)]mean = [shape * scale].

In case of `impairment_type_index != DELAY`, (1) mean plus 4 times standard deviation should be less than or equal to max allowed(4194288). (2)shape and scale should be greater than or equal to 0.

In case of `impairment_type_index = DELAY`, mean plus 4 times standard deviation should be less than or equal to the maximum latency.

Actions

set, get

Parameters

1. `shape`: *long integer*, specifies the shape. Units: none. Default value: 0.
2. `scale`: *long integer*, specifies the Gamma function scale parameter.

Example

```
# set
input:  0/1 PED_GAMMA [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_GAMMA [0, 0] ?
output: 0/1 PED_GAMMA [0, 0] 1 1
```

PED_GAUSS

`code:  1628`

```
# set
<module-index>/<port-index> PED_GAUSS [<flow_index>, <impairment_type_
↪index>] <mean> <std_deviation>

# get
<module-index>/<port-index> PED_GAUSS [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Gaussian distribution.

Note: In case of `impairment_type_index != DELAY`: (1) mean plus 3 times standard deviation should be less than or equal to max allowed (4194288). (2) mean should always be at least 3 times the standard deviation, this to ensure that the impairment distance is always positive.

In case of `impairment_type_index = DELAY`: (1) mean plus 3 times standard deviation should be less than or equal to the maximum latency. (2) mean minus 3 times the standard deviation should be greater than or equal to minimum latency.

Actions

set, get

Parameters

1. `mean`: *long integer*, specifies the Gaussian mean.
2. `std_deviation`: *long integer*, specifies the Gaussian standard deviation.

Example

```
# set
input:  0/1 PED_GAUSS [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_GAUSS [0, 0] ?
output: 0/1 PED_GAUSS [0, 0] 1 1
```

PED_GE

`code:  1626`

```
# set
<module-index>/<port-index> PED_GE [<flow_index>, <impairment_type_
↪index>] <goodprob> <goodtransprob> <badprob> <badtransprob>

# get
<module-index>/<port-index> PED_GE [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Gilbert-Elliot distribution.

Actions

set, get

Parameters

1. `goodprob`: *integer*, specifies the good state probability in ppm. Default value: 0.
2. `goodtransprob`: *integer*, specifies the good state transition probability in ppm. Default value: 0.
3. `badprob`: *integer*, specifies the bad state probability in ppm. Default value: 0.
4. `badtransprob`: *integer*, specifies the bad state transition probability in ppm. Default value: 0.

Example

```
# set
input:  0/1 PED_GE [0, 0] 1 1 1 1
output: <OK>

# get
input:  0/1 PED_GE [0, 0] ?
output: 0/1 PED_GE [0, 0] 1 1 1 1
```

PED_OFF

`code:  1620`

```
# set
<module-index>/<port-index> PED_OFF [<flow_index>, <impairment_type_
↪index>]
```

Description

Configure Impairments Distribution to OFF. Assigning a different distribution than OFF to an impairment will activate the impairment. To de-activate the impairment assign distribution OFF.

Actions

set

Parameters

Example

```
# set
input:  0/1 PED_OFF [0, 0]
output: <OK>
```

PED_ONESHOTSTATUS

code: 1612

```
# get
<module-index>/<port-index> PED_ONESHOTSTATUS [<flow_index>,
↪<impairment_type_index>] ?
```

Description

Retrieves the one-shot completion status.

Note: The return value is only valid, if the configured distribution is either accumulate & burst (DELAY) or fixed burst (non-DELAY).

Actions

get

Parameters

1. one_shot_status: *short integer*, the one-shot completion status

Example

```
# get
input:  0/1 PED_ONESHOTSTATUS [0, 0] ?
output: 0/1 PED_ONESHOTSTATUS [0, 0] 123
```

PED_POISSON

code: 1629

```
# set
<module-index>/<port-index> PED_POISSON [<flow_index>, <impairment_
↪type_index>] <mean>

# get
<module-index>/<port-index> PED_POISSON [<flow_index>, <impairment_
↪type_index>] ?
```

Description

Configuration of "Poisson" distribution.

Note: Standard deviation is derived from mean, i.e., standard deviation = SQRT(mean).

In case of impairment_type_index != DELAY, mean plus 3 times standard deviation should be less than or equal to max allowed (4194288).

In case of impairment_type_index = DELAY, mean plus 3 times standard deviation should be less than or equal to the maximum latency.

Actions

set, get

Parameters

1. mean: *long integer*, specifies the Poisson mean value.

Example

```
# set
input:  0/1 PED_POISSON [0, 0] 1
output: <OK>

# get
input:  0/1 PED_POISSON [0, 0] ?
output: 0/1 PED_POISSON [0, 0] 1
```

PED_RANDOM

code: 1622

```
# set
<module-index>/<port-index> PED_RANDOM [<flow_index>, <impairment_type_
↪index>] <probability>

# get
<module-index>/<port-index> PED_RANDOM [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Random Rate distribution. Packets are impaired randomly based on a per packet probability. This way the impaired fraction of packets will be equal to the configured probability over time. Random probability in ppm (i.e. 1 means 0.0001%)

Actions

set, get

Parameters

1. `probability`: *integer*, specifies the random probability in ppm. Default value is 0.

Example

```
# set
input:  0/1 PED_RANDOM [0, 0] 1
output: <OK>

# get
input:  0/1 PED_RANDOM [0, 0] ?
output: 0/1 PED_RANDOM [0, 0] 1
```

PED_RANDOMBURST

code: 1625

```
# set
<module-index>/<port-index> PED_RANDOMBURST [<flow_index>, <impairment_
↪type_index>] <minimum> <maximum> <probability>

# get
<module-index>/<port-index> PED_RANDOMBURST [<flow_index>, <impairment_
↪type_index>] ?
```

Description

Configuration of Random Burst distribution.

Actions

set, get

Parameters

1. `minimum`: *integer*, specifies minimum burst size. Default value: 0 (Range 0 to 65535)
2. `maximum`: *integer*, specifies maximum burst size. Default value: 0 (Range 0 to 65535)
3. `probability`: *integer*, specifies the per packet probability of initiating a burst in ppm. Default value: 0.

Example

```
# set
input:  0/1 PED_RANDOMBURST [0, 0] 1 1 1
output: <OK>

# get
input:  0/1 PED_RANDOMBURST [0, 0] ?
output: 0/1 PED_RANDOMBURST [0, 0] 1 1 1
```

PED_SCHEDULE

`code:  1611`

```
# set
<module-index>/<port-index> PED_SCHEDULE [<flow_index>, <impairment_
↪type_index>] <duration> <period>

# get
<module-index>/<port-index> PED_SCHEDULE [<flow_index>, <impairment_
↪type_index>] ?
```

Description

Configure the impairment scheduler function. The configuration of the scheduler depends on the type of distribution to schedule:

(1) Burst distributions: “Fixed Burst” and “Accumulate and Burst”.

(2) Non-Burst distributions: All others. For burst distributions, the scheduler can be configured for “One-shot” operation or “Repeat Operation”. When running in “Repeat Operation” the “Repeat Period” must be configured. For non-burst distributions, the scheduler can be configured

operate in either "Continuous" or "Repeat Period" modes. When running in "Repeat Period" configuration of "Duration" and "Repeat Period" is required.

Actions

set, get

Parameters

1. `duration`: *integer*, specifies the "on" period. Units = multiples of 10 ms (range 1 to 65535), default is 1
2. `period`: *integer*, specifies the "total" period. Units = multiples of 10 ms (range 0 to 65535), default is 0

Example

```
# set
input:  0/1 PED_SCHEDULE [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_SCHEDULE [0, 0] ?
output: 0/1 PED_SCHEDULE [0, 0] 1 1
```

PED_STEP

`code:  1642`

```
# set
<module-index>/<port-index> PED_STEP [<flow_index>, <impairment_type_
↪index>] <low> <high>

# get
<module-index>/<port-index> PED_STEP [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Step distribution (DELAY only).

Note: If the low/high is less than minimum latency, value is set to minimum latency. If the low/high is greater than maximum latency, value is set to maximum latency.

Actions

set, get

Parameters

1. `low`: *long integer*, specifies the packet delay in the 'low' state of the step. Units = nanosecond (must be multiples of 100 ns).
2. `high`: *long integer*, specifies the packet delay in the 'high' state of the step. Units = nanosecond (must be multiples of 100 ns).

Example

```
# set
input:  0/1 PED_STEP [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_STEP [0, 0] ?
output: 0/1 PED_STEP [0, 0] 1 1
```

PED_UNI

`code:  1627`

```
# set
<module-index>/<port-index> PED_UNI [<flow_index>, <impairment_type_
↪index>] <minimum> <maximum>

# get
<module-index>/<port-index> PED_UNI [<flow_index>, <impairment_type_
↪index>] ?
```

Description

Configuration of Uniform distribution.

Note: If minimum is less than minimum latency, value is set to minimum latency. If minimum is greater than maximum latency, value is set to maximum latency.

Actions

set, get

Parameters

1. `minimum`: *long integer*, in case of iid != DELAY, specifies the minimum no. of packets. Default value: 0 (Range 0 to 4194288). In case of iid = DELAY, specifies the minimum latency limit. Unit is nanosecond (must be multiples of 100 ns). Default value: minimum latency.
2. `maximum`: *long integer*, in case of iid != DELAY, specifies the maximum no. of packets. Default value: 0 (Range 0 to 4194288). In case of iid = DELAY, specifies the maximum latency limit. Unit is nanosecond (must be multiples of 100 ns). Default value: minimum latency.

Example

```
# set
input:  0/1 PED_UNI [0, 0] 1 1
output: <OK>

# get
input:  0/1 PED_UNI [0, 0] ?
output: 0/1 PED_UNI [0, 0] 1 1
```

Custom Distribution

PEC_COMMENT

`code:  1681`

```
# set
<module-index>/<port-index> PEC_COMMENT [<custom_distribution_index>]
```

(continues on next page)

(continued from previous page)

```
↪<comment>

# get
<module-index>/<port-index> PEC_COMMENT [<custom_distribution_index>] ?
```

Description

Defines the user-defined description string of a custom distribution.

Actions

set, get

Parameters

1. `comment`: *string*, the user-specified comment/description for the custom distribution.

Example

```
# set
input:  0/1 PEC_COMMENT [0] 'A String'
output: <OK>

# get
input:  0/1 PEC_COMMENT [0] ?
output: 0/1 PEC_COMMENT [0] 'A String'
```

PEC_DELETE

`code:  1682`

```
# set
<module-index>/<port-index> PEC_DELETE [<custom_distribution_index>]
```

Description

Deletes the custom distribution definition.

Note: Once a customer has defined a customer distribution using PEC_VAL, it is defined until it is explicitly deleted. Only customer distributions which are not referenced by any impairments, can be deleted.

Actions

set

Parameters

Example

```
# set
input:  0/1 PEC_DELETE [0]
output: <OK>
```

PEC_DISTTYPE

code: 1683

```
# get
<module-index>/<port-index> PEC_DISTTYPE [<custom_distribution_index>]␣
↪?
```

Description

Retrieves if a custom distribution is defined for latency or non-latency.

Note: Using PEC_DISTTYPE as set has no effect. The disttype is determined upon custom distribution creation and cannot be modified later. However, it is legal to issue the PEC_DISTTYPE set command with no effect.

Actions

get

Parameters

1. `latency_type`: *byte*, latency type of a custom distribution

- `INTERPACKET_DISTRIBUTION = 0`
- `LATENCY_DISTRIBUTION = 1`

Example

```
# get
input:  0/1 PEC_DISTTYPE [0] ?
output: 0/1 PEC_DISTTYPE [0] INTERPACKET_DISTRIBUTION
```

PEC_INDICES

`code:  1610`

```
# set
<module-index>/<port-index> PEC_INDICES <indices>

# get
<module-index>/<port-index> PEC_INDICES ?
```

Description

The full list of which custom distributions which are defined for a port. These are the custom id values that are used for assigning the custom distributions to an impairment. Setting the value of this command creates a new custom distribution (default values) for each value that is not already in use, and deletes each custom distribution that is not mentioned in the list. The same can be accomplished one-custom-distribution-at-a-time using the `PEC_VAL` and `PEC_DELETE` commands.

Note: Custom distributions which are currently defined are not affected when mentioned in a `PEC_INDICES` set command. Custom distributions which are currently assigned to an impairment cannot be deleted and any attempt of deleting such a custom distribution using either `PEC_DELETE` or `PEC_INDICES` will result in an error.

Actions

set, get

Parameters

1. `indices`: *integer list*, a list of indices to create new custom distributions

Example

```
# set
input:  0/1 PEC_INDICES 0 1
output: <OK>

# get
input:  0/1 PEC_INDICES ?
output: 0/1 PEC_INDICES 0 1
```

PEC_VAL

`code:  1680`

```
# set
<module-index>/<port-index> PEC_VAL [<custom_distribution_index>]
 ↪<linear> <symmetric> <entry_count> <data_x>

# get
<module-index>/<port-index> PEC_VAL [<custom_distribution_index>] ?
```

Description

Definition of custom distribution. Custom distributions can be defined for latency with 1024 entries and for non-latency impairments with 512 entries. Each port will maintain a list of defined custom distributions, identified by an CUST_ID. (Range: 1 - 40).

Actions

set, get

Parameters

1. `linear`: *byte*, defines the way the FPGA RAM content is played out
   - `OFF = 0`
   - `ON = 1`
2. `symmetric`: *byte*, reserved for future use, must be set to 0.
   - `OFF = 0`
   - `ON = 1`
3. `entry_count`: *integer*, defines the number of entries in `data_x` (allowed value: 512,1024). For Latency, 1024 entries are used, and for rest, 512 entries are used.
4. `data_x`: *long integer list*, array size equals to `entry_count`, holds values to be filled in the RAM memory.

Example

```
# set
input:  0/1 PEC_VAL [0] OFF OFF 2 0 1
output: <OK>

# get
input:  0/1 PEC_VAL [0] ?
output: 0/1 PEC_VAL [0] OFF OFF 2 0 1
```

Filter

There are 2 register copies used to configure the filters:

(1) `Shadow-copy (type value = 0)`, temporary copy configured by sever. Values stored in `shadow-copy` have no immediate effect on the flow filters. `PEF_APPLY` will pass the values from the `shadow-copy` to the `working-copy`.

(2) `Working-copy (type value = 1)`, reflects what is currently used for filtering in the FPGA. `Working-copy` cannot be written directly. Only `shadow-copy` allows direct write.

(3) All `set` actions are performed on `shadow-copy` ONLY.

(4) Only when `PEF_APPLY` is called, `working-copy` and FPGA are updated with values from the `shadow-copy`.

Note: Flow filter is only applicable to flow ID from 1 to 7. You cannot place a filter on flow 0.

PEF_ANYCONFIG

code: 1729

```
# set
<module-index>/<port-index> PEF_ANYCONFIG [<flow_index>, <filter_type>
↪] <position> <value> <mask>

# get
<module-index>/<port-index> PEF_ANYCONFIG [<flow_index>, <filter_type>
↪] ?
```

Description

Basic mode only. Defines the ANY field filter configuration. The "ANY field" filter will match 6 consecutive bytes in the incoming packets at a programmable offset. Applying a mask, allows to only filter based on selected bits within the 6 bytes.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `position`: *short integer*, specifies the start position of the ANY field. Default value: 0, Range:0-127
2. `value`: *hex6*, specifying the six bytes of the field. Default value: 0x000000000000
3. `mask`: *hex6*, specifying the six bytes of the field. Default value: 0xFFFFFFFFFFFF

Example

```
# set
input:  0/1 PEF_ANYCONFIG [1,0] 0 0x000000000000 0xFFFFFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_ANYCONFIG [1,0] ?
output: 0/1 PEF_ANYCONFIG [1,0] 0 0x000000000000 0xFFFFFFFFFFFF
```

PEF_ANYSETTINGS

code: 1728

```
# set
<module-index>/<port-index> PEF_ANYSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_ANYSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines if filtering on ANY field in a packet is used for flow filtering.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of ANY field information.
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of ANY field information.
   - `EXCLUDE = 0`

- INCLUDE = 1

Example

```
# set
input:  0/1 PEF_ANYSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_ANYSETTINGS [1,0] ?
output: 0/1 PEF_ANYSETTINGS [1,0] OFF EXCLUDE
```

PEF_APPLY

code: 1701

```
# set
<module-index>/<port-index> PEF_APPLY [<flow_index>, <filter_type>]
```

Description

Applies filter definitions from *shadow-copy* to *working-copy*. This also pushes these settings to the FPGA.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PEF_APPLY [1]
output: <OK>
```

PEF_CANCEL

code: 1735

```
# set
<module-index>/<port-index> PEF_CANCEL [<flow_index>]
```

Description

Undo updates to shadow filter settings, sets dirty false.

Actions

set

Parameters

Example

```
# set
input:  0/1 PEF_CANCEL [1]
output: <OK>
```

PEF_CONFIG

code: 1606

```
# get
<module-index>/<port-index> PEF_CONFIG [<flow_index>, <filter_type>] ?
```

Description

Get filter config

Actions

get

Parameters

Example

```
# get
input:  0/1 PEF_CONFIG [1,0] ?
```

PEF_ENABLE

code: 1702

```
# set
<module-index>/<port-index> PEF_ENABLE [<flow_index>, <filter_type>]
 ↪<state>

# get
<module-index>/<port-index> PEF_ENABLE [<flow_index>, <filter_type>] ?
```

Description

Defines if filtering is enabled for the flow.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `state`: *byte*, state of the filter
   - `OFF = 0`
   - `ON = 1`

Example

```
# set
input:  0/1 PEF_ENABLE [1,0] OFF
output: <OK>

# get
input:  0/1 PEF_ENABLE [1,0] ?
output: 0/1 PEF_ENABLE [1,0] OFF
```

PEF_ETHDESTADDR

`code:  1705`

```
# set
<module-index>/<port-index> PEF_ETHDESTADDR [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_ETHDESTADDR [<flow_index>, <filter_
↪type>] ?
```

Description

Defines the Ethernet Destination Address settings for the Ethernet filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of Ethernet Destination Address information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *hex6*, specifying the six bytes of the address. Default value: 0x000000000000
3. `mask`: *hex6*, specifying the mask corresponding to the address. Default value: 0xFFFFFFFFFFFF

Example

```
# set
input:  0/1 PEF_ETHDESTADDR [1,0] OFF 0x000000000000 0xFFFFFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_ETHDESTADDR [1,0] ?
output: 0/1 PEF_ETHDESTADDR [1,0] OFF 0x000000000000 0xFFFFFFFFFFFF
```

PEF_ETHSETTINGS

`code:  1703`

```
# set
<module-index>/<port-index> PEF_ETHSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_ETHSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Defines what filter action is performed on the Ethernet header.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies if Ethernet information is expected.
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the use of Ethernet information.
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_ETHSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_ETHSETTINGS [1,0] ?
output: 0/1 PEF_ETHSETTINGS [1,0] OFF EXCLUDE
```

PEF_ETHSRCADDR

`code:  1704`

```
# set
<module-index>/<port-index> PEF_ETHSRCADDR [<flow_index>, <filter_type>
↪] <use> <value> <mask>

# get
```

(continues on next page)

(continued from previous page)

```
<module-index>/<port-index> PEF_ETHSRCADDR [<flow_index>, <filter_type>
↪] ?
```

Description

Defines the Ethernet Source Address settings for the Ethernet filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of Ethernet Source Address information.
   - `OFF = 0`
   - `ON = 1`
2. `value`: *hex6*, specifying the six bytes of the address. Default value: 0x000000000000.
3. `mask`: *hex6*, specifying the mask corresponding to the address. Default value: 0xFFFFFFFFFFFF.

Example

```
# set
input:  0/1 PEF_ETHSRCADDR [1,0] OFF 0x000000000000 0xFFFFFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_ETHSRCADDR [1,0] ?
output: 0/1 PEF_ETHSRCADDR [1,0] OFF 0x000000000000 0xFFFFFFFFFFFF
```

PEF_INIT

code: 1700

```
# set
<module-index>/<port-index> PEF_INIT [<flow_index>]
```

Description

Prepares for setting up a filter definition. When called, all filter definitions in the *shadow-copy*, which are not applied, are discarded and replaced with the default values.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PEF_INIT [1]
output: <OK>
```

PEF_IPV4DESTADDR

code: 1716

```
# set
<module-index>/<port-index> PEF_IPV4DESTADDR [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV4DESTADDR [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines the IPv4 Destination Address settings for the IPv4 filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv4 Destination Address information.
   - `OFF = 0`
   - `ON = 1`
2. `value`: *address*, specifying the four bytes of the address. Default value: 0.0.0.0
3. `mask`: *hex4*, specifying the filter mask of the value. Default value: 0xFFFFFFFF

Example

```
# set
input:  0/1 PEF_IPV4DESTADDR [1,0] OFF 192.168.1.100 0xFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_IPV4DESTADDR [1,0] ?
output: 0/1 PEF_IPV4DESTADDR [1,0] OFF 192.168.1.100 0xFFFFFFFF
```

PEF_IPV4DSCP

`code:  1717`

```
# set
<module-index>/<port-index> PEF_IPV4DSCP [<flow_index>, <filter_type>]
 ↪<use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV4DSCP [<flow_index>, <filter_type>]␣
 ↪?
```

Description

Basic mode only. Defines if IPv4 DSCP/TOS settings used for the IPv4 filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv4 DSCP/TOS information.
   - `OFF = 0`
   - `ON = 1`
2. `value`: *short integer*, specifying the value of the IPv4 DSCP/TOS in the upper 6 bits. value[7:2] = DSCP/TOS, value[1:0] = reserved (must be zero). Default vaule: 0
3. `mask`: *hex*, specifying the filter mask of the value in the upper 6 bits. mask[7:2] = DSCP/TOS mask, mask[1:0] = reserved (must be zero). Default value: 0xFC

Example

```
# set
input:  0/1 PEF_IPV4DSCP [1,0] OFF 0 0xFC
output: <OK>

# get
input:  0/1 PEF_IPV4DSCP [1,0] ?
output: 0/1 PEF_IPV4DSCP [1,0] OFF 0 0xFC
```

PEF_IPV4SETTINGS

code: 1714

```
# set
<module-index>/<port-index> PEF_IPV4SETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
```

(continues on next page)

(continued from previous page)

```
<module-index>/<port-index> PEF_IPV4SETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines what filter action is performed on the IPv4 header.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv4 information
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of IPv4 information
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_IPV4SETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_IPV4SETTINGS [1,0] ?
output: 0/1 PEF_IPV4SETTINGS [1,0] OFF EXCLUDE
```

PEF_IPV4SRCADDR

code: 1715

```
# set
<module-index>/<port-index> PEF_IPV4SRCADDR [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV4SRCADDR [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines the IPv4 Source Address settings for the IPv4 filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv4 Source Address information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *address*, specifying the four bytes of the address. Default value: 0.0.0.0
3. `mask`: *hex4*, specifying the filter mask of the value. Default value: 0xFFFFFFFF

Example

```
# set
input:  0/1 PEF_IPV4SRCADDR [1,0] OFF 192.168.1.100 0xFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_IPV4SRCADDR [1,0] ?
output: 0/1 PEF_IPV4SRCADDR [1,0] OFF 192.168.1.100 0xFFFFFFFF
```

PEF_IPV6DESTADDR

code: 1720

```
# set
<module-index>/<port-index> PEF_IPV6DESTADDR [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV6DESTADDR [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines the IPv6 Destination Address settings for the IPv6 filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv6 Destination Address information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *string*, specifying the address. Default : `0x00000000000000000000000000000000`
3. `mask`: `hex16`, specifying the six first bytes of the address. Default value: `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF`

Example

```
# set
input: 0/1 PEF_IPV6DESTADDR [1,0] OFF␣
↪0x00000000000000000000000000000000 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
output: <OK>
```

(continues on next page)

(continued from previous page)

```
# get
input:  0/1 PEF_IPV6DESTADDR [1,0] ?
output: 0/1 PEF_IPV6DESTADDR [1,0] OFF␣
↪0x00000000000000000000000000000000 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

PEF_IPV6SETTINGS

code: 1718

```
# set
<module-index>/<port-index> PEF_IPV6SETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_IPV6SETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines what filter action is performed on the IPv6 header.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv6 header
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of IPv6 header
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_IPV6SETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_IPV6SETTINGS [1,0] ?
output: 0/1 PEF_IPV6SETTINGS [1,0] OFF EXCLUDE
```

PEF_IPV6SRCADDR

code: 1719

```
# set
<module-index>/<port-index> PEF_IPV6SRCADDR [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV6SRCADDR [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines the IPv6 Source Address settings for the IPv6 filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of IPv6 Source Address information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *string*, specifying the address. Default : `0x00000000000000000000000000000000`

3. `mask:` `hex16`, specifying the six first bytes of the address. Default value: `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF`

Example

```
# set
input:  0/1 PEF_IPV6SRCADDR [1,0] OFF␣
↪0x00000000000000000000000000000000 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_IPV6SRCADDR [1,0] ?
output: 0/1 PEF_IPV6SRCADDR [1,0] OFF␣
↪0x00000000000000000000000000000000 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

PEF_IPV6TC

`code:  1721`

```
# set
<module-index>/<port-index> PEF_IPV6TC [<flow_index>, <filter_type>]
↪<use> <value> <mask>

# get
<module-index>/<port-index> PEF_IPV6TC [<flow_index>, <filter_type>] ?
```

Description

Basic mode only. Defines the IPv6 Traffic Class settings used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of the IPv6 Traffic Class information.
   - `OFF = 0`
   - `ON = 1`
2. `value`: *string*, specifying the value of the IPv6 Traffic Class in the upper 6 bits. value[7:2] = IPv6 Traffic Class. value[1:0] = reserved (must be zero). Default value: 0
3. `mask`: *hex*, specifying the filter mask for the value in the upper 6 bits. mask[7:2] = IPv6 Traffic Class mask. mask[1:0] = reserved (must be zero). Default value: 0xFC

Example

```
# set
input:  0/1 PEF_IPV6TC [1,0] OFF 0 0xFC
output: <OK>

# get
input:  0/1 PEF_IPV6TC [1,0] ?
output: 0/1 PEF_IPV6TC [1,0] OFF 0 0xFC
```

PEF_ISSHADOWDIRTY

`code:  1734`

```
# get
<module-index>/<port-index> PEF_ISSHADOWDIRTY [<flow_index>] ?
```

Description

Get shadow filter status (if shadow is in sync with working copy or not).

Actions

get

Parameters

Example

```
# get
input:  0/1 PEF_ISSHADOWDIRTY [1] ?
output: 0/1 PEF_ISSHADOWDIRTY [1] 0
```

PEF_L2PUSE

code: 1706

```
# set
<module-index>/<port-index> PEF_L2PUSE [<flow_index>, <filter_type>]
 ↪<use>

# get
<module-index>/<port-index> PEF_L2PUSE [<flow_index>, <filter_type>] ?
```

Description

Defines what Layer 2+ protocols that are present and may be used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the presence of Layer 2+ protocols.

- `NA = 0`
- `VLAN1 = 1`
- `VLAN2 = 2`
- `MPLS = 3`

Example

```
# set
input:  0/1 PEF_L2PUSE [1,0] NA
output: <OK>

# get
input:  0/1 PEF_L2PUSE [1,0] ?
output: 0/1 PEF_L2PUSE [1,0] NA
```

PEF_L3USE

code: 1713

```
# set
<module-index>/<port-index> PEF_L3USE [<flow_index>, <filter_type>]
↪<use>

# get
<module-index>/<port-index> PEF_L3USE [<flow_index>, <filter_type>] ?
```

Description

Basic mode only. Defines what Layer 3 protocols that are present and may be used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the presence of Layer 3 protocols

- `NA = 0`
- `IP4 = 1`
- `IP6 = 2`

Example

```
# set
input:  0/1 PEF_L3USE [1,0] IP4
output: <OK>

# get
input:  0/1 PEF_L3USE [1,0] ?
output: 0/1 PEF_L3USE [1,0] IP4
```

PEF_MASK

code: 1778

```
# set
<module-index>/<port-index> PEF_MASK [<flow_index>, <filter_type>,
↪<protocol_segment_index>] <masks>

# get
<module-index>/<port-index> PEF_MASK [<flow_index>, <filter_type>,
↪<protocol_segment_index>] ?
```

Description

This command is valid only for `Extended filter mode` (check `PEF_MODE`.).

Defines the mask byte values that select the values specified by `PEF_VALUE`.

For a chosen `<protocol_segment_index>` the first byte in the value masks the first byte of the corresponding `PEF_VALUE`, and so on.

If `<protocol_segment_index> = 0`, the maximum number of match value bytes that can be set is determined by the total length of the protocol segments specified with `PEF_PROTOCOL`. E.g. if `PEF_PROTOCOL` is set to ETHERNET then only 12 bytes can be set. In order to set the full 128 bytes, either specify a detailed protocol segment list, or use the raw protocol segment type. This specifies 12 + 116 = 128 bytes.

If `<protocol_segment_index> != 0`, only the bytes covered by that segment are manipulated, so if `PEF_PROTOCOL` is set to `ETHERNET VLAN ETHERTYPE ECPRI`, then `<protocol_segment_index> = 4` selects the 8 bytes of the eCPRI header starting at byte position (12 + 2 + 4) = 18.

For `set` command where fewer value bytes are provided than specified by the protocol segment, those unspecified bytes are set to zero.

The `get` command always returns the number of bytes specified by the protocol segment.

Actions

set, get

Parameters

1. `masks`: *hex list*, mask byte values

Example

```
# set
input:  0/1 PEF_MASK [1,0,1] 0xFFFFFFFFFFFFFFFFFFFFFFFF
output: <OK>

# get
input:  0/1 PEF_MASK [1,0,1] ?
output: 0/1 PEF_MASK [1,0,1] 0xFFFFFFFFFFFFFFFFFFFFFFFF
```

PEF_MODE

`code:  1780`

```
# set
<module-index>/<port-index> PEF_MODE [<flow_index>, <filter_type>]
 ↪<mode>

# get
<module-index>/<port-index> PEF_MODE [<flow_index>, <filter_type>] ?
```

Description

Control the filter mode.

Actions

set, get

Parameters

1. `mode:` `FilterMode`, the mode of the filter.

- `BASIC = 0`
- `EXTENDED = 1`

Example

```
# set
input:  0/1 PEF_MODE [1,0] BASIC
output: <OK>

# get
input:  0/1 PEF_MODE [1,0] ?
output: 0/1 PEF_MODE [1,0] BASIC
```

PEF_MPLSLABEL

`code:` `1711`

```
# set
<module-index>/<port-index> PEF_MPLSLABEL [<flow_index>, <filter_type>
↪] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_MPLSLABEL [<flow_index>, <filter_type>
↪] ?
```

Description

Basic mode only. Defines the MPLS label settings for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of MPLS label information.
   - `OFF = 0`
   - `ON = 1`
2. `value`: *integer*, specifying the 20-bit value of the label. Default value: 0.
3. `mask`: *hex3*, specifying the 20-bit value of the label. Default value: 0x0FFFFF

Example

```
# set
input:  0/1 PEF_MPLSLABEL [1,0] OFF 0 0x0FFFFF
output: <OK>

# get
input:  0/1 PEF_MPLSLABEL [1,0] ?
output: 0/1 PEF_MPLSLABEL [1,0] OFF 0 0x0FFFFF
```

PEF_MPLSSETTINGS

`code:  1710`

```
# set
<module-index>/<port-index> PEF_MPLSSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_MPLSSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines what filter action is performed on the MPLS header.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of MPLS information
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies specifies if MPLS information is expected
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_MPLSSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_MPLSSETTINGS [1,0] ?
output: 0/1 PEF_MPLSSETTINGS [1,0] OFF EXCLUDE
```

PEF_MPLSTOC

`code:  1712`

```
# set
<module-index>/<port-index> PEF_MPLSTOC [<flow_index>, <filter_type>]
 ↪<use> <value> <mask>

# get
<module-index>/<port-index> PEF_MPLSTOC [<flow_index>, <filter_type>] ?
```

Description

Basic mode only. Defines the MPLS TOC settings for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of MPLS TOC information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *short integer*, specifying the value of the MPLS TOC. Default value: 0 (Range: 0 to 7).
3. `mask`: *hex*, specifying the filter mask for the value of the MPLS TOC. Default value: 0x07

Example

```
# set
input:  0/1 PEF_MPLSTOC [1,0] OFF 0 0x07
output: <OK>

# get
input:  0/1 PEF_MPLSTOC [1,0] ?
output: 0/1 PEF_MPLSTOC [1,0] OFF 0 0x07
```

PEF_PROTOCOL

`code:  1779`

```
# set
<module-index>/<port-index> PEF_PROTOCOL [<flow_index>, <filter_type>]
 ↪<segment_list>

# get
<module-index>/<port-index> PEF_PROTOCOL [<flow_index>, <filter_type>]␣
 ↪?
```

Description

Extended mode only. Defines the sequence of protocol segments that can be matched. The total length of the specified segments cannot exceed 128 bytes. If an existing sequence of segments is changed (using PEF_PROTOCOL) the underlying value and mask bytes remain unchanged, even though the semantics of those bytes may have changed. However, if the total length, in bytes, of the segments is reduced, then the excess bytes of value and mask are set to zero. I.e. to update an existing filter, you must first correct the list of segments (using PEF_PROTOCOL) and subsequently update the filtering value (using PEF_VALUE) and filtering mask (PEF_MASK).

Actions

set, get

Parameters

1. segment_list: *byte list*, specifying the list of protocol segment types in the order they are expected in a frame. First segment type must be ETHERNET; the following can be chosen freely.

- ETHERNET = 1
- VLAN = 2
- ARP = 3
- IP = 4
- IPV6 = 5
- UDP = 6
- TCP = 7
- LLC = 8
- SNAP = 9
- GTP = 10
- ICMP = 11
- RTP = 12
- RTCP = 13
- STP = 14
- SCTP = 15
- MACCTRL = 16
- MPLS = 17
- PBBTAG = 18

- FCOE = 19
- FC = 20
- FCOETAIL = 21
- IGMPV3L0 = 22
- IGMPV3L1 = 23
- UDPCHECK = 24
- IGMPV2 = 25
- MPLS_TP_OAM = 26
- GRE_NOCHECK = 27
- GRE_CHECK = 28
- TCPCHECK = 29
- GTPV1L0 = 30
- GTPV1L1 = 31
- GTPV2L0 = 32
- GTPV2L1 = 33
- IGMPV1 = 34
- PWETHCTRL = 35
- VXLAN = 36
- ETHERNET_8023 = 37
- NVGRE = 38
- DHCPV4 = 39
- GENEVE = 40
- XENA_TPLD = 41
- XENA_TPLD_PI = 42
- XENA_MICROTPLD = 43
- ETHERNET_FCS = 44
- MACCTRLPFC = 45
- ECPRI = 46
- ROE = 47
- ETHERTYPE = 48
- -n (n bytes custom segment)

Example

```
# set
input:  0/1 PEF_PROTOCOL [1,0] ETHERNET VLAN ECPRI
output: <OK>

# get
input:  0/1 PEF_PROTOCOL [1,0] ?
output: 0/1 PEF_PROTOCOL [1,0] ETHERNET VLAN ECPRI
```

PEF_TCPDESTPORT

code: 1727

```
# set
<module-index>/<port-index> PEF_TCPDESTPORT [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_TCPDESTPORT [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines TCP Destination Port settings used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of TCP Destination Port information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *integer*, specifies the value of the TCP Destination Port. Default value: 0
3. `mask`: *hex2*, specifies the filter mask for the value. Default value: 0xFFFF

Example

```
# set
input:  0/1 PEF_TCPDESTPORT [1,0] OFF 0 0xFFFF
output: <OK>

# get
input:  0/1 PEF_TCPDESTPORT [1,0] ?
output: 0/1 PEF_TCPDESTPORT [1,0] OFF 0 0xFFFF
```

PEF_TCPSETTINGS

code: 1725

```
# set
<module-index>/<port-index> PEF_TCPSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_TCPSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines if filtering on TCP information is used for flow filtering.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of TCP information.
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of TCP information.
   - `EXCLUDE = 0`

- INCLUDE = 1

Example

```
# set
input:  0/1 PEF_TCPSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_TCPSETTINGS [1,0] ?
output: 0/1 PEF_TCPSETTINGS [1,0] OFF EXCLUDE
```

PEF_TCPSRCPORT

code: 1726

```
# set
<module-index>/<port-index> PEF_TCPSRCPORT [<flow_index>, <filter_type>
↪] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_TCPSRCPORT [<flow_index>, <filter_type>
↪] ?
```

Description

Basic mode only. Defines TCP Source Port settings used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. use: *byte*, specifies the use of TCP Source Port information
   - OFF = 0
   - ON = 1
2. value: *integer*, specifies the value of the TCP Source Port. Default value: 0
3. mask: *hex2*, specifies the filter mask for the value. Default value: 0xFFFF

Example

```
# set
input:  0/1 PEF_TCPSRCPORT [1,0] OFF 0 0xFFFF
output: <OK>

# get
input:  0/1 PEF_TCPSRCPORT [1,0] ?
output: 0/1 PEF_TCPSRCPORT [1,0] OFF 0 0xFFFF
```

PEF_TPLDCONFIG

code: 1731

```
# set
<module-index>/<port-index> PEF_TPLDCONFIG [<flow_index>, <filter_type>
↪, <test_payload_filter_index>] <use> <id>

# get
<module-index>/<port-index> PEF_TPLDCONFIG [<flow_index>, <filter_type>
↪, <test_payload_filter_index>] ?
```

Description

Defines the TPLD filter configuration. There are only 16 TPLD filter, thus <test_payload_filter_index> is from 0 to 15.

Note: For set, the only allowed filter_type is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of TPLD field information
   - `OFF = 0`
   - `ON = 1`
2. `id`: *integer*, specifies the TPLD ID. Range: 0-2015, Default value: 0

Example

```
# set
input:  0/1 PEF_TPLDCONFIG [1,0,0] OFF 0
output: <OK>

# get
input:  0/1 PEF_TPLDCONFIG [1,0,0] ?
output: 0/1 PEF_TPLDCONFIG [1,0,0] OFF 0
```

PEF_TPLDSETTINGS

`code:  1730`

```
# set
<module-index>/<port-index> PEF_TPLDSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_TPLDSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Defines if filtering on TPLD field in a packet is used for flow filtering. The TPLD filter allows filtering based on the Xena Test payload ID. The Test payload ID is meta data, which can be inserted into the Ethernet packets by Xena traffic generators. For each flow filter, can the filter be based on 16 TPLD ID values.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of TPLD information.
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of TPLD information.
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_TPLDSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_TPLDSETTINGS [1,0] ?
output: 0/1 PEF_TPLDSETTINGS [1,0] OFF EXCLUDE
```

PEF_UDPDESTPORT

`code:  1724`

```
# set
<module-index>/<port-index> PEF_UDPDESTPORT [<flow_index>, <filter_
↪type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_UDPDESTPORT [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Defines UDP Destination Port settings used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*. .

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of UDP Destination Port information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *integer*, specifying the value of the UDP Destination Port. Default value: 0
3. `mask`: *hex2*, specifying the filter mask for the value. Default value: 0xFFFF

Example

```
# set
input:  0/1 PEF_UDPDESTPORT [1,0] OFF 0 0xFFFF
output: <OK>

# get
input:  0/1 PEF_UDPDESTPORT [1,0] ?
output: 0/1 PEF_UDPDESTPORT [1,0] OFF 0 0xFFFF
```

PEF_UDPSETTINGS

`code:  1722`

```
# set
<module-index>/<port-index> PEF_UDPSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_UDPSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Basic mode only. Controls if UDP packet information is used for flow filtering.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of UDP information.
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of UDP information.
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_UDPSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_UDPSETTINGS [1,0] ?
output: 0/1 PEF_UDPSETTINGS [1,0] OFF EXCLUDE
```

PEF_UDPSRCPORT

`code:  1723`

```
# set
<module-index>/<port-index> PEF_UDPSRCPORT [<flow_index>, <filter_type>
↪] <use> <value> <mask>

# get
```

(continues on next page)

(continued from previous page)

```
<module-index>/<port-index> PEF_UDPSRCPORT [<flow_index>, <filter_type>
↪] ?
```

Description

Basic mode only. Defines UDP Source Port settings used for the filter.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of UDP Source Port information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *integer*, specifying the value of the UDP Source Port. Default value: 0
3. `mask`: *hex2*, specifying the filter mask for the value. Default value: 0xFFFF

Example

```
# set
input:  0/1 PEF_UDPSRCPORT [1,0] OFF 0 0xFFFF
output: <OK>

# get
input:  0/1 PEF_UDPSRCPORT [1,0] ?
output: 0/1 PEF_UDPSRCPORT [1,0] OFF 0 0xFFFF
```

PEF_VALUE

code: 1777

```
# set
<module-index>/<port-index> PEF_VALUE [<flow_index>, <filter_type>,
↪<protocol_segment_index>] <pid> <value>

# get
<module-index>/<port-index> PEF_VALUE [<flow_index>, <filter_type>,
↪<protocol_segment_index>] ?
```

Description

This command is valid only for `Extended filter mode` (check `PEF_MODE`.).

Defines the byte values that can be matched if selected by `PEF_MASK`.

If `<protocol_segment_index> = 0`, the maximum number of match value bytes that can be set is determined by the total length of the protocol segments specified with `PEF_PROTOCOL`. E.g. if `PEF_PROTOCOL` is set to ETHERNET then only 12 bytes can be set. In order to set the full 128 bytes, either specify a detailed protocol segment list, or use the raw protocol segment type. This specifies 12 + 116 = 128 bytes.

If `<protocol_segment_index> != 0`, only the bytes covered by that segment are manipulated, so if `PEF_PROTOCOL` is set to `ETHERNET VLAN ETHERTYPE ECPRI`, then `<protocol_segment_index> = 4` selects the 8 bytes of the eCPRI header starting at byte position (12 + 2 + 4) = 18.

For `set` command where fewer value bytes are provided than specified by the protocol segment, those unspecified bytes are set to zero.

The `get` command always returns the number of bytes specified by the protocol segment.

Actions

set, get

Parameters

1. `value`: *hex list*, the raw bytes comprising the packet header

Example

```
# set
input:  0/1 PEF_VALUE [1,0,1] 0x00000000000000000000000000
output: <OK>

# get
input:  0/1 PEF_VALUE [1,0,1] ?
output: 0/1 PEF_VALUE [1,0,1] 0x00000000000000000000000000
```

PEF_VLANPCP

code: 1709

```
# set
<module-index>/<port-index> PEF_VLANPCP [<flow_index>, <filter_type>,
↪<vlan_type>] <use> <value> <mask>

# get
<module-index>/<port-index> PEF_VLANPCP [<flow_index>, <filter_type>,
↪<vlan_type>] ?
```

Description

Basic mode only. Defines the VLAN PCP settings for the VLAN filter.

<vlan_type> specifies the VLAN type:

- VLAN1 (0), the inner VLAN Tag is specified for the filter (used also when there is only one VLAN tag) indicates single/inner VLAN-TPID = 0x8100.
- VLAN2 (1), the outer VLAN Tag is specified for the filter indicates outer VLAN-TPID=0x88A8.

Note: For set, the only allowed filter_type is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies the use of VLAN PCP information
   - `OFF = 0`
   - `ON = 1`
2. `value`: *short integer*, specifying the value of the PCP. Default value: 0 (Range: 0 to 7)
3. `mask`: *hex*, specifying the 8 bit value mask. Default value: 0x07

Example

```
# set
input:  0/1 PEF_VLANPCP [1,0,0] OFF 0 0x07
output: <OK>

# get
input:  0/1 PEF_VLANPCP [1,0,0] ?
output: 0/1 PEF_VLANPCP [1,0,0] OFF 0 0x07
```

PEF_VLANSETTINGS

code: 1707

```
# set
<module-index>/<port-index> PEF_VLANSETTINGS [<flow_index>, <filter_
↪type>] <use> <action>

# get
<module-index>/<port-index> PEF_VLANSETTINGS [<flow_index>, <filter_
↪type>] ?
```

Description

Defines what filter action is performed on the VLAN header.

Note: For `set`, the only allowed `filter_type` is *shadow-copy*.

Actions

set, get

Parameters

1. `use`: *byte*, specifies if VLAN information is expected
   - `OFF = 0`
   - `AND = 1`
2. `action`: *byte*, specifies the action of VLAN information
   - `EXCLUDE = 0`
   - `INCLUDE = 1`

Example

```
# set
input:  0/1 PEF_VLANSETTINGS [1,0] OFF EXCLUDE
output: <OK>

# get
input:  0/1 PEF_VLANSETTINGS [1,0] ?
output: 0/1 PEF_VLANSETTINGS [1,0] OFF EXCLUDE
```

PEF_VLANTAG

`code:  1708`

```
# set
<module-index>/<port-index> PEF_VLANTAG [<flow_index>, <filter_type>,
↪<vlan_type>] <use> <value> <mask>

# get
```

(continues on next page)

(continued from previous page)

```
<module-index>/<port-index> PEF_VLANTAG [<flow_index>, <filter_type>,
↪<vlan_type>] ?
```

Description

Basic mode only. Defines the VLAN TAG settings for the VLAN filter.

<vlan_type> specifies the VLAN type:

- VLAN1 (0), the inner VLAN Tag is specified for the filter (used also when there is only one VLAN tag) indicates single/inner VLAN-TPID = 0x8100.
- VLAN2 (1), the outer VLAN Tag is specified for the filter indicates outer VLAN-TPID=0x88A8.

Note: For set, the only allowed filter_type is *shadow-copy*.

Actions

set, get

Parameters

1. use: *byte*, specifies the use of VLAN TAG information
   - OFF = 0
   - ON = 1
2. value: *integer*, specifying the 12 bit value of the tag. Default value: 0.
3. mask: *hex2*, specifying the 12 bit value of the tag. Default value: 0x0FFF

Example

```
# set
input:  0/1 PEF_VLANTAG [1,0,0] OFF 0 0x0FFF
output: <OK>

# get
input:  0/1 PEF_VLANTAG [1,0,0] ?
output: 0/1 PEF_VLANTAG [1,0,0] OFF 0 0x0FFF
```

Impairment Statistics

Impairment flow-level statistics commands.

PE_FLOWCLEAR

code: 1776

```
# set
<module-index>/<port-index> PE_FLOWCLEAR [<flow_index>]
```

Description

Clear all the impairment (duplicate, drop, mis-ordered, corrupted, latency and jitter) statistics on a particular flow on the port. The byte and packet counts will restart at zero.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PE_FLOWCLEAR [0]
output: <OK>
```

PE_FLOWCORTOTAL

code: 1774

```
# get
<module-index>/<port-index> PE_FLOWCORTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets corrupted in a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `total_corrupted_pkt_count`: *long integer*, number of packets corrupted for the flow,
2. `fcs_corrupted_pkt_count`: *long integer*, number of packets with Ethernet FCS corrupted for the flow,
3. `ip_corrupted_pkt_count`: *long integer*, number of packets with IP header checksum corrupted for the flow,
4. `udp_corrupted_pkt_count`: *long integer*, number of packets with UDP checksum corrupted for the flow,
5. `tcp_corrupted_pkt_count`: *long integer*, number of packets with TCP checksum corrupted for the flow,
6. `total_corrupted_pkt_ratio`: *long integer*, ratio of number of packets corrupted for the flow expressed in ppm,
7. `fcs_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with Ethernet FCS corrupted for the flow expressed in ppm,
8. `ip_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with IP Header checksum corrupted for the flow expressed in ppm,
9. `udp_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with UDP checksum corrupted for the flow expressed in ppm,
10. `tcp_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with TCP checksum corrupted for the flow expressed in ppm.

Example

```
# get
input:  0/1 PE_FLOWCORTOTAL [0] ?
output: 0/1 PE_FLOWCORTOTAL [0] 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

PE_FLOWDROPTOTAL

code: 1770

```
# get
<module-index>/<port-index> PE_FLOWDROPTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets dropped in a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_drop_count_total`: *long integer*, total number of packets dropped for the flow,
2. `pkt_drop_count_programmed`: *long integer*, total number of packets dropped as programmed for the flow,
3. `pkt_drop_count_bandwidth`: *long integer*, total number of packets dropped due to bandwidth control for the flow,
4. `pkt_drop_count_other`: *long integer*, total number of packets dropped for other reasons for the flow,
5. `pkt_drop_ratio_total`: *long integer*, ratio of number of packets dropped for the flow, expressed in ppm,
6. `pkt_drop_ratio_programmed`: *long integer*, ratio of number of packets dropped as programmed for the flow, expressed in ppm,
7. `pkt_drop_ratio_bandwidth`: *long integer*, ratio of number of packets dropped due to bandwidth control for the flow, expressed in ppm,
8. `pkt_drop_ratio_other`: *long integer*, ratio of number of packets dropped for other reasons for the flow, expressed in ppm.

Example

```
# get
input:  0/1 PE_FLOWDROPTOTAL [0] ?
output: 0/1 PE_FLOWDROPTOTAL [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

PE_FLOWDUPTOTAL

code: 1772

```
# get
<module-index>/<port-index> PE_FLOWDUPTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets duplicated in a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets duplicated for the flow, ratio of number of packets duplicated for the flow expressed in ppm.
2. `ratio`: *long integer*, number of packets duplicated for the flow, ratio of number of packets duplicated for the flow expressed in ppm.

Example

```
# get
input:  0/1 PE_FLOWDUPTOTAL [0] ?
output: 0/1 PE_FLOWDUPTOTAL [0] 123456789123 123456789123
```

PE_FLOWJITTERTOTAL

code: 1775

```
# get
<module-index>/<port-index> PE_FLOWJITTERTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets jittered in a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets jittered in the flow, ratio of number of packets jittered in the flow, expressed in ppm
2. `ratio`: *long integer*, number of packets jittered in the flow, ratio of number of packets jittered in the flow, expressed in ppm

Example

```
# get
input:  0/1 PE_FLOWJITTERTOTAL [0] ?
output: 0/1 PE_FLOWJITTERTOTAL [0] 123456789123 123456789123
```

PE_FLOWLATENCYTOTAL

code: 1771

```
# get
<module-index>/<port-index> PE_FLOWLATENCYTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets delayed between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets delayed in the flow, ratio of number of packets delayed in the flow, expressed in ppm.
2. `ratio`: *long integer*, number of packets delayed in the flow, ratio of number of packets delayed in the flow, expressed in ppm.

Example

```
# get
input:  0/1 PE_FLOWLATENCYTOTAL [0] ?
output: 0/1 PE_FLOWLATENCYTOTAL [0] 123456789123 123456789123
```

PE_FLOWMISTOTAL

`code:  1773`

```
# get
<module-index>/<port-index> PE_FLOWMISTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets mis-ordered in a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets mis-ordered for the flow, ratio of number of packets, expressed in ppm.
2. `ratio`: *long integer*, number of packets mis-ordered for the flow, ratio of number of packets, expressed in ppm.

Example

```
# get
input:  0/1 PE_FLOWMISTOTAL [0] ?
output: 0/1 PE_FLOWMISTOTAL [0] 123456789123 123456789123
```

PT_FLOWCLEAR

`code:  1743`

```
# set
<module-index>/<port-index> PT_FLOWCLEAR [<flow_index>]
```

Description

Clear all the transmit statistics on a particular flow for a E100 Chimera port. The byte and packet counts will restart at zero.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PT_FLOWCLEAR [0]
output: <OK>
```

PT_FLOWTOTAL

code: 1740

```
# get
<module-index>/<port-index> PT_FLOWTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets transmitted from a between this receive port and its partner TX port.

Actions

get

Parameters

1. `l2_bps`: *long integer*, number of bits transmitted at layer 2 in the last second for the flow,
2. `pps`: *long integer*, number of packets transmitted in the last second for the flow,
3. `byte_count`: *long integer*, number of bytes transmitted since statistics were cleared for the flow,
4. `packet_count`: *long integer*, number of packets transmitted since statistics were cleared for the flow

Example

```
# get
input:  0/1 PT_FLOWTOTAL [0] ?
output: 0/1 PT_FLOWTOTAL [0] 123456789123 123456789123 123456789123␣
↪123456789123
```

PR_FLOWCLEAR

code: 1742

```
# set
<module-index>/<port-index> PR_FLOWCLEAR [<flow_index>]
```

Description

Clear all the receive statistics on a particular flow for a E100 Chimera port. The byte and packet counts will restart at zero.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PR_FLOWCLEAR [0]
output: <OK>
```

PR_FLOWTOTAL

code: 1741

```
# get
<module-index>/<port-index> PR_FLOWTOTAL [<flow_index>] ?
```

Description

Obtains statistics concerning all the packets received from a flow between this receive port and its partner TX port.

Actions

get

Parameters

1. `l2_bps`: *long integer*, number of bits received at layer 2 in the last second for the flow,
2. `pps`: *long integer*, number of packets received in the last second for the flow,
3. `byte_count`: *long integer*, number of bytes received since statistics were cleared for the flow,
4. `packet_count`: *long integer*, number of packets received since statistics were cleared for the flow

Example

```
# get
input:  0/1 PR_FLOWTOTAL [0] ?
output: 0/1 PR_FLOWTOTAL [0] 123456789123 123456789123 123456789123␣
↪123456789123
```

Impairment port-level statistics commands.

PE_CLEAR

code: 1756

```
# set
<module-index>/<port-index> PE_CLEAR
```

Description

Clear all the impairment (duplicate, drop, mis-ordered, corrupted, latency and jitter) statistics for a E100 Chimera port and flows on the port. The byte and packet counts will restart at zero.

Actions

set

Parameters

None

Example

```
# set
input:  0/1 PE_CLEAR
output: <OK>
```

PE_CORTOTAL

code: 1754

```
# get
<module-index>/<port-index> PE_CORTOTAL ?
```

Description

Obtains statistics concerning all the packets corrupted on between this receive port and its partner TX port.

Actions

get

Parameters

1. `total_corrupted_pkt_count`: *long integer*, number of packets corrupted in all flows;
2. `fcs_corrupted_pkt_count`: *long integer*, number of packets with Ethernet FCS corrupted in all flows;
3. `ip_corrupted_pkt_count`: *long integer*, number of packets with IP header checksum corrupted in all flows;
4. `udp_corrupted_pkt_count`: *long integer*, number of packets with UDP checksum corrupted in all flows;
5. `tcp_corrupted_pkt_count`: *long integer*, number of packets with TCP checksum corrupted in all flows;
6. `total_corrupted_pkt_ratio`: *long integer*, ratio of number of packets corrupted in all flows, expressed in ppm;
7. `fcs_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with Ethernet FCS corrupted in all flows expressed in ppm;
8. `ip_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with IP Header checksum corrupted in all flows, expressed in ppm;
9. `udp_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with UDP checksum corrupted in all flows, expressed in ppm;
10. `tcp_corrupted_pkt_ratio`: *long integer*, ratio of number of packets with TCP checksum corrupted in all flows, expressed in ppm

Example

```
# get
input:  0/1 PE_CORTOTAL ?
output: 0/1 PE_CORTOTAL 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

PE_DROPTOTAL

code: 1750

```
# get
<module-index>/<port-index> PE_DROPTOTAL ?
```

Description

Obtains statistics concerning all the packets dropped between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_drop_count_total`: *long integer*, total number of packets dropped in all flows.
2. `pkt_drop_count_programmed`: *long integer*, total number of packets dropped as programmed in all flows.
3. `pkt_drop_count_bandwidth`: *long integer*, total number of packets dropped due to bandwidth control in all flows.
4. `pkt_drop_count_other`: *long integer*, total number of packets dropped for other reasons in all flows.
5. `pkt_drop_ratio_total`: *long integer*, ratio of number of packets dropped in all flows, expressed in ppm.
6. `pkt_drop_ratio_programmed`: *long integer*, ratio of number of packets dropped as programmed in all flows, expressed in ppm.
7. `pkt_drop_ratio_bandwidth`: *long integer*, ratio of number of packets dropped due to bandwidth control in all flows, expressed in ppm.
8. `pkt_drop_ratio_other`: *long integer*, ratio of number of packets dropped for other reasons in all flows, expressed in ppm.

Example

```
# get
input:  0/1 PE_DROPTOTAL ?
output: 0/1 PE_DROPTOTAL 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123
```

PE_DUPTOTAL

code: 1752

```
# get
<module-index>/<port-index> PE_DUPTOTAL ?
```

Description

Obtains statistics concerning all the packets duplicated between this receive port and its partner TX port.

Actions

get

Parameters

1. *pkt_count* `: *long integer*, number of packets duplicated in all flows, ratio of number of packets duplicated in all flows, expressed in ppm.
2. `ratio`: *long integer*, number of packets duplicated in all flows, ratio of number of packets duplicated in all flows, expressed in ppm.

Example

```
# get
input:  0/1 PE_DUPTOTAL ?
output: 0/1 PE_DUPTOTAL 123456789123 123456789123
```

PE_JITTERTOTAL

code: 1755

```
# get
<module-index>/<port-index> PE_JITTERTOTAL ?
```

Description

Obtains statistics concerning all the packets jittered between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets jittered in all flows, ratio of number of packets jittered in all flows expressed in ppm.
2. `ratio`: *long integer*, number of packets jittered in all flows, ratio of number of packets jittered in all flows expressed in ppm.

Example

```
# get
input:  0/1 PE_JITTERTOTAL ?
output: 0/1 PE_JITTERTOTAL 123456789123 123456789123
```

PE_LATENCYTOTAL

code: 1751

```
# get
<module-index>/<port-index> PE_LATENCYTOTAL ?
```

Description

Obtains statistics concerning all the packets delayed this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets delayed in all flows, and ratio of number of packets delayed in all flows, expressed in ppm.
2. `ratio`: *long integer*, number of packets delayed in all flows, and ratio of number of packets delayed in all flows, expressed in ppm.

Example

```
# get
input:  0/1 PE_LATENCYTOTAL ?
output: 0/1 PE_LATENCYTOTAL 123456789123 123456789123
```

PE_MISTOTAL

`code:  1753`

```
# get
<module-index>/<port-index> PE_MISTOTAL ?
```

Description

Obtains statistics concerning all the packets mis-ordered between this receive port and its partner TX port.

Actions

get

Parameters

1. `pkt_count`: *long integer*, number of packets mis-ordered in all flows, number of packets mis-ordered in all flows, expressed in ppm.
2. `ratio`: *long integer*, number of packets mis-ordered in all flows, number of packets mis-ordered in all flows, expressed in ppm.

Example

```
# get
input:  0/1 PE_MISTOTAL ?
output: 0/1 PE_MISTOTAL 123456789123 123456789123
```

3.3.5 Config

P_CONFIG

`code:  131`

```
# get
<module-index>/<port-index> P_CONFIG ?
```

Description

Return port configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 P_CONFIG ?
```

3.3.6 Full Config

P_FULLCONFIG

code: 130

```
# get
<module-index>/<port-index> P_FULLCONFIG ?
```

Description

Return port full configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 P_FULLCONFIG ?
```

PC_FULLCONFIG

code: 136

```
# get
<module-index>/<port-index> PC_FULLCONFIG ?
```

Description

Return port capture configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 PC_FULLCONFIG ?
```

PD_FULLCONFIG

code: 137

```
# get
<module-index>/<port-index> PD_FULLCONFIG ?
```

Description

Return port histogram configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 PD_FULLCONFIG ?
```

PE_FULLCONFIG

code: 1607

```
# get
<module-index>/<port-index> PE_FULLCONFIG ?
```

Description

Return flow config for all the flows of a port

Actions

get

Parameters

Example

```
# get
input: 0/1 PE_FULLCONFIG ?
```

PEC_FULLCONFIG

code: 1609

```
# get
<module-index>/<port-index> PEC_FULLCONFIG ?
```

Description

Return custom distribution config for a port

Actions

get

Parameters

Example

```
# get
input:  0/1 PEC_FULLCONFIG ?
```

PEF_FULLCONFIG

code: 1733

```
# get
<module-index>/<port-index> PEF_FULLCONFIG ?
```

Description

Return filter config for all the flows off a port

Actions

get

Parameters

Example

```
# get
input:  0/1 PEF_FULLCONFIG ?
```

PF_FULLCONFIG

code: 135

```
# get
<module-index>/<port-index> PF_FULLCONFIG ?
```

Description

Return port filter configuration

Actions

get

Parameters

Example

```
# get
input: 0/1 PF_FULLCONFIG ?
```

PL_FULLCONFIG

code: 134

```
# get
<module-index>/<port-index> PL_FULLCONFIG ?
```

Description

Return port length term configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 PL_FULLCONFIG ?
```

PM_FULLCONFIG

code: 133

```
# get
<module-index>/<port-index> PM_FULLCONFIG ?
```

Description

Return port match term configuration

Actions

get

Parameters

Example

```
# get
input:  0/1 PM_FULLCONFIG ?
```

PS_FULLCONFIG

code: 132

```
# get
<module-index>/<port-index> PS_FULLCONFIG ?
```

Description

Return port stream full configuration

Actions

get

Parameters

Example

```
# get
input: 0/1 PS_FULLCONFIG ?
```

3.4 Misc

This section describes the CLI for Xena's stateful TGA Vulcan.

As an alternative to using the application VulcanManager, you can interact with the testers using **XOA CLI for Vulcan**. This also allows the tester to be controlled from a scripting environment, and be part of a larger automation environment.

Commands are logically grouped in a hierarchy. At the top level we have a Chassis. Currently there are two different Vulcan testers: VulcanBay and VulcanCompact. For VulcanBay and VulcanCompact, the entire chassis is considered one *L47 Module*, however in the future a chassis may have several Vulcan modules.

A Vulcan Module has several *ports* (currently between 1 and 12). Like on Valkyrie, each port must be reserved before it can be configured and traffic can be started, allowing multiple users to work with the Vulcan product at the same time.

In addition to ports a Vulcan Module contains a number of *Packet Engines (PE)* , which generates and handles the TCP traffic. As default each port is allocated one PE, however more PEs can be allocated to a Vulcan port increasing the performance on that port. Packet Engines are a shared resource between the Vulcan ports on a Vulcan module. Currently VulcanCompact contains 5 PEs and VulcanBay contains 2 groups of 14 PEs.

The *Stream* concept in Valkyrie has been replaced by *Connection Groups (CG)* in Vulcan. A *CG* specifies a number of TCP connections (1 to 2 million per PE). Several Connection Groups can be configured on a Port (currently up to 200).

A *CG* has a configured *Load Profile*, which defines the ramp-up start time along with the durations of the ramp-up, steady-state and ramp-down periods. A *CG* is configured with an *Application Type* and an *Application Scenario*. The Application Type defines the type of data transmitted by the TCP connections, and the Application Scenario defines the data flow between Servers and Clients.

By combining several Connection Groups on a port, it is possible to create a mixture of different traffic types and scenarios, and to create complex resulting load profiles.

3.4.1 Vulcan Port States

Vulcan test ports have seven states: `OFF`, `PREPARE`, `PREPARE_RDY`, `PRERUN`, `PRERUN_RDY`, `RUNNING` and `STOPPED`. Traffic is generated in the `PRERUN` and `RUNNING` states only, and configuration of parameters is only valid in state `OFF` except for a few runtime options.

Port traffic commands can be set with `P4_TRAFFIC` and port state queried by `P4_STATE`.

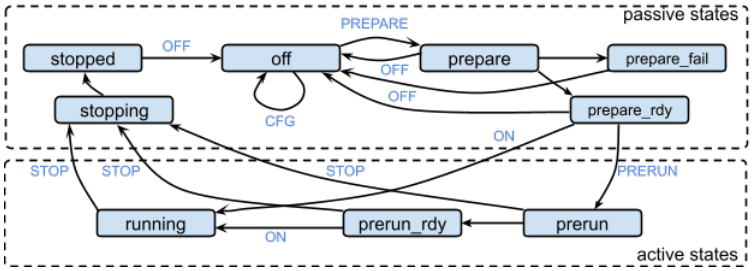


Fig. 3.16: L47 Port State Diagram

- `OFF` - default state. Entered from `STOPPED` or `PREPARE` on `OFF` command. This is the only state that allows configuration commands. `P4_RESET` is also considered a configuration command. Upon entering `OFF` state, some internal "house cleaning" is done. For example: freeing TCP Connections, clearing test specific counters etc.
- `PREPARE` - this state is entered from state `OFF` on `PREPARE` command. Here internal data structures relevant for the test configuration are created. When done the state changes to `PREPARE_RDY` or PREPARE_FAIL and, a `P4_STATE PREPARE_RDY` or `P4_STATE PREPARE_FAIL` notification is sent to all users logged on to the chassis.
- `PREPARE_RDY` - entered automatically after activities in `PREPARE` have completed successfully.
- `PREPARE_FAIL` - entered automatically from `PREPARE`, if an error occurs. An error could for example be failure to load a configured replay_file.
- `PRERUN` - entered from `PREPARE_RDY` on `PRERUN` command. If enabled, this is where ARP and NDP requests are sent. When done the state changes to `PRERUN_RDY` and a `P4_STATE PRERUN_RDY` notification is sent to all users logged on to the chassis.
- `PRERUN_RDY` - entered automatically after activities in `PRERUN` have completed.

- `RUNNING` - entered either from `PREPARE_RDY` or `PRERUN_RDY` on `ON` command. This is where TCP connections are established, payload is generated and connections are closed again.
- `STOPPING` - entered from `RUNNING`, `PRERUN_RDY` or `PRERUN` on `STOP` command. Stops Rx/Tx traffic. In the `STOPPING` state, post-test data are calculated and captured packets are saved to files.
- `STOPPED` - entered automatically after activities in `STOPPING` are complete. This is where we can read post-test statistics and extract captured packets.

3.4.2 CLI for Vulcan

CLI commands for Vulcan functionalities:

Vulcan Module

This module contains the **Vulcan module commands**.

M4_CAPTURE_FILE_DELETE

```
# set
<module-index> M4_CAPTURE_FILE_DELETE <filename>
```

Description

Command to delete a Capture File (`.pcap` file) in the Capture File directory (`/var/ftp/pub/replay/pcap/`). For information about the location and directory structure for the Capture Files, see: M4_REPLAY_PARSE_START

Actions

set

Parameters

`filename`: *string*, file name (including relative path and excluding the `.pcap` extension).

Example

```
# set
input:  0 M4_CAPTURE_FILE_DELETE 'A String'
output: <OK>
```

M4_CAPTURE_FILE_LIST

```
# get
<module-index> M4_CAPTURE_FILE_LIST ?
```

Description

Returns a list of Capture Files (`.pcap` files) in the 'user' Capture File directory (`/var/ftp/pub/replay/pcap/user/`).

Actions

get

Parameters

`file_list`: *string*, list of Capture Files in BSON document on the tester

Example

```
# get
input:  0 M4_CAPTURE_FILE_LIST ?
output: 0 M4_CAPTURE_FILE_LIST 'A String'
```

M4_CAPTURE_FILE_LIST_BSON

```
# get
<module-index> M4_CAPTURE_FILE_LIST_BSON ?
```

Description

Works as M4_CAPTURE_FILE_LIST, but returns the file list formatted as a BSON document.

Actions

get

Parameters

bson: *short integer list*, the capture file list in BSON document

Example

```
# get
input:  0 M4_CAPTURE_FILE_LIST_BSON ?
output: 0 M4_CAPTURE_FILE_LIST_BSON 123 123
```

M4_CAPTURE_SIZE

```
# set
<module-index> M4_CAPTURE_SIZE <size>

# get
<module-index> M4_CAPTURE_SIZE ?
```

Description

Specify whether to capture whole packets(large) or truncated packets. When truncated (small) is selected only the first 128 bytes of the packet are saved.

Actions

set, get

Parameters

size: *byte*, specifying whether to capture whole packets or truncated packets.

- FULL = 0
- SMALL = 1

Example

```
# set
input:  0 M4_CAPTURE_SIZE FULL
output: <OK>

# get
input:  0 M4_CAPTURE_SIZE ?
output: 0 M4_CAPTURE_SIZE FULL
```

M4_COMPATIBLE_CLIENT_VERSION ==========^^^^

```
# get
<module-index> M4_COMPATIBLE_CLIENT_VERSION ?
```

Description

Returns the recommended and required VulcanMananger client version.

Actions

get

Parameters

recommended_major: *integer*, the recommended and required VulcanMananger client version.

recommended_minor: *integer*, the recommended and required VulcanMananger client version.

recommended_minor_2: *integer*, the recommended and required VulcanMananger client version.

recommended_minor_3: *integer*, the recommended and required VulcanMananger client version.

required_major: *integer*, the recommended and required VulcanMananger client version.

required_minor: *integer*, the recommended and required VulcanMananger client version.

`required_minor_2`: *integer*, the recommended and required VulcanMananger client version.

`required_minor_3`: *integer*, the recommended and required VulcanMananger client version.

Example

```
# get
input:  0 M4_COMPATIBLE_CLIENT_VERSION ?
output: 0 M4_COMPATIBLE_CLIENT_VERSION 1 1 1 1 1 1 1 1
```

M4_LICENSE_INFO

```
# get
<module-index> M4_LICENSE_INFO ?
```

Description

Returns the number of available and free PE licenses. Only 'available' number of PEs can simultaneously be assigned to reserved ports.

Actions

get

Parameters

`pes_available`: *integer*, the number of available and free PE licenses

`pes_free`: *integer*, the number of available and free PE licenses

`N1g_available`: *integer*, the number of available and free PE licenses

`N1g_free`: *integer*, the number of available and free PE licenses

`N10g_available`: *integer*, the number of available and free PE licenses

`N10g_free`: *integer*, the number of available and free PE licenses

`N25g_available`: *integer*, the number of available and free PE licenses

`N25g_free`: *integer*, the number of available and free PE licenses

`N40g_available`: *integer*, the number of available and free PE licenses

`N40g_free`: *integer*, the number of available and free PE licenses

Example

```
# get
input:  0 M4_LICENSE_INFO ?
output: 0 M4_LICENSE_INFO 28 20 4 2 0 0 4 4 0 0
```

M4_MEM_INFO

```
# get
<module-index> M4_MEM_INFO ?
```

Description

Return the system memory information.

Actions

get

Parameters

`year`: *long integer*, the system memory information.

`month`: *long integer*, the system memory information.

Example

```
# get
input:  0 M4_MEM_INFO ?
output: 0 M4_MEM_INFO 123456789123 123456789123
```

M4_REPLAY_FILE_DELETE

```
# set
<module-index> M4_REPLAY_FILE_DELETE <filename>
```

Description

Command to delete a Replay File (`.bson` file) in the Replay File directory (`/var/ftp/pub/replay/bson/`). For information about the location and directory structure for the Replay Files, see: M4_REPLAY_PARSE_START

Actions

set

Parameters

`filename`: *string*, file name (including relative path and excluding the `.bson` extension).

Example

```
# set
input:  0 M4_REPLAY_FILE_DELETE 'A String'
output: <OK>
```

M4_REPLAY_FILE_LIST

```
# get
<module-index> M4_REPLAY_FILE_LIST ?
```

Description

Returns a list of Replay Files (`.bson` files) in the 'user' Replay File directory (`/var/ftp/pub/replay/bson/user/`).

Actions

get

Parameters

`file_list`: *string*, a list of Replay Files in BSON document on the tester

Example

```
# get
input:  0 M4_REPLAY_FILE_LIST ?
output: 0 M4_REPLAY_FILE_LIST 'A String'
```

M4_REPLAY_FILE_LIST_BSON

```
# get
<module-index> M4_REPLAY_FILE_LIST_BSON ?
```

Description

Works as `M4_REPLAY_FILE_LIST`, but returns the file list formatted as a BSON document.

Actions

get

Parameters

`bson`: *short integer list*, the replay file list in BSON format

Example

```
# get
input:  0 M4_REPLAY_FILE_LIST_BSON ?
output: 0 M4_REPLAY_FILE_LIST_BSON 123 123
```

M4_REPLAY_PARSE_START

```
# set
<module-index> M4_REPLAY_PARSE_START <filename>
```

Description

Command to start parsing an uploaded Capture File (in PCAP format) intended for use in a replay test scenario. The result of the parsing - if successful - is a Replay File (in BSON format) with the same name as the Capture File, which can be used as parameter to P4G_REPLAY_filename command. If parsing is unsuccessful, a Replay File is created containing the parse result. The M4_REPLAY_FILE_INFO_BSON command can be used to get information about a Replay File - including the parse result. PCAP Capture Files can be uploaded to the L47 chassis using FTP. The 'root' location of Capture Files uploaded manually by the user is /var/ftp/pub/replay/pcap/. Three subdirectories exist: cache/, user/ and xena/. cache / and xena/ is used by Vulcan Manager, and user/ is intended for manual upload and parsing of Capture Files. A similar directory structure is present for Replay Files generated by the parsing, and the 'root' location is /var/ftp/pub/replay/bson/.

Actions

set

Parameters

`filename`: *string*, filename (including relative path and excluding the '.pcap' extension).

Example

```
# set
input:  0 M4_REPLAY_PARSE_START 'A String'
output: <OK>
```

M4_REPLAY_PARSE_STATE

```
# get
<module-index> M4_REPLAY_PARSE_STATE ?
```

Description

Only one Capture File can be parsed at a time. This command returns the state of the parser, which can be PARSING or OFF. M4_REPLAY_PARSE_START command is only accepted when the parser state is OFF.

Actions

get

Parameters

state: *byte*, **capture file parsing state**

- OFF = 0
- PARSING = 1

Example

```
# get
input:  0 M4_REPLAY_PARSE_STATE ?
output: 0 M4_REPLAY_PARSE_STATE OFF
```

M4_REPLAY_PARSE_STOP

```
# set
<module-index> M4_REPLAY_PARSE_STOP
```

Description

Command to stop parsing a Capture File. Parsing of very large Capture Files may take several seconds, and may be aborted using this command. No parameters

Actions

set

Parameters

Example

```
# set
input:  0 M4_REPLAY_PARSE_STOP
output: <OK>
```

M4_REPLAY_PARSER_PARAMS

```
# set
<module-index> M4_REPLAY_PARSER_PARAMS <tcp_port>

# get
<module-index> M4_REPLAY_PARSER_PARAMS ?
```

Description

Configuration of parameters for the parsing of pcap files.

Actions

set, get

Parameters

`tcp_port`: *integer*, server-side TCP port of the dummy TCP connection inserted in UDP.

Example

```
# set
input:  0 M4_REPLAY_PARSER_PARAMS 1
output: <OK>

# get
```

(continues on next page)

(continued from previous page)

```
input:  0 M4_REPLAY_PARSER_PARAMS ?
output: 0 M4_REPLAY_PARSER_PARAMS 1
```

M4_SYSTEM_STATUS

```
# get
<module-index> M4_SYSTEM_STATUS ?
```

Description

Returns the L47 module system status in a text string.

Actions

get

Parameters

`status_string`: *string*, the L47 module system status in a text string

Example

```
# get
input:  0 M4_SYSTEM_STATUS ?
output: 0 M4_SYSTEM_STATUS "OK"
```

M4_SYSTEM_TIME

```
# set
<module-index> M4_SYSTEM_TIME <year> <month> <day> <hour> <minute>
 ↪<second>

# get
<module-index> M4_SYSTEM_TIME ?
```

Description

Sets or returns the modules system time in UTC.

Actions

set, get

Parameters

`year`: *integer*, the year

`month`: *integer*, the month

`day`: *integer*, the day of the month

`hour`: *integer*, the hour

`minute`: *integer*, the minute

`second`: *integer*, the second

Example

```
# set
input:  0 M4_SYSTEM_TIME 2020 4 19 7 25 00
output: <OK>

# get
input:  0 M4_SYSTEM_TIME ?
output: 0 M4_SYSTEM_TIME 2020 4 19 7 25 00
```

M4_SYSTEMID

```
# get
<module-index> M4_SYSTEMID ?
```

Description

Return the system identifier of a L47 module.

Actions

get

Parameters

`system_id`: *string*, the system identifier of a L47 module.

Example

```
# get
input:  0 M4_SYSTEMID ?
output: 0 M4_SYSTEMID
 ↪"7ea0f7d001cea4e79a110aa8c78222fd12665326acbcaf13282f37472bc596ec"
```

M4_TIME

```
# get
<module-index> M4_TIME ?
```

Description

Returns the module time in millisecond.

Actions

get

Parameters

`time_now`: *long integer*, the module time in millisecond.

Example

```
# get
input:  0 M4_TIME ?
output: 0 M4_TIME 123456789123
```

M4_TLS_CIPHER_SUITES

```
# get
<module-index> M4_TLS_CIPHER_SUITES ?
```

Description

Returns a list of supported TLS Cipher Suites.

Actions

get

Parameters

`cipher_suites`: *hex list*, list of IANA values of supported cipher suites

Example

```
# get
input:  0 M4_TLS_CIPHER_SUITES ?
output: 0 M4_TLS_CIPHER_SUITES 0x57
```

M4_VERSIONNO

```
# get
<module-index> M4_VERSIONNO ?
```

Description

Returns a version string containing a combination of information regarding the software version and the build environment. The first part of the string is the software build version.

Actions

get

Parameters

`version_string`: *string*, a version string containing a combination of information regarding the software version and the build environment

Example

```
# get
input:  0 M4_VERSIONNO ?
output: 0 M4_VERSIONNO "0.1.0 2014-12-17-055000[xena47hp:cu][3.14.4-
↪200.fc20.x86_64]c95923b"
```

Vulcan Packet Engine

This module contains the **Vulcan module-level packet engine commands**.

M4E_MODE

```
# set
<module-index> M4E_MODE <mode>

# get
<module-index> M4E_MODE ?
```

Description

Select resource allocation mode.

Actions

set, get

Parameters

mode: *byte*, **resource allocation mode**

- SIMPLE = 0
- ADVANCED = 1

Example

```
# set
input:  0 M4E_MODE SIMPLE
output: <OK>

# get
input:  0 M4E_MODE ?
output: 0 M4E_MODE SIMPLE
```

M4E_RESERVE

```
# set
<module-index> M4E_RESERVE <mask>

# get
<module-index> M4E_RESERVE ?
```

Description

Advanced mode only: Reserve a number of PEs so they later can be assigned to specific ports.

Actions

set, get

Parameters

`mask`: *hex8*, bitmask of PEs to reserve

Example

```
# set
input:  0 M4E_RESERVE 0x00000000000000ff
output: <OK>

# get
input:  0 M4E_RESERVE ?
output: 0 M4E_RESERVE 0x00000000000000ff
```

This module contains the **Vulcan port-level packet engine commands**.

P4E_ALLOCATE

```
# set
<module-index> P4E_ALLOCATE <pe_count_alloc>

# get
<module-index> P4E_ALLOCATE ?
```

Description

Simple mode only: Allocate a number of PEs to this port.

Actions

set, get

Parameters

`pe_count_alloc`: *integer*, the total number of PEs to allocate to this port - including the PEs already allocated to the port.

Example

```
# set
input:  1/1 P4E_ALLOCATE 2
output: <OK>

# get
input:  1/1 P4E_ALLOCATE ?
output: 1/1 P4E_ALLOCATE 2
```

P4E_ALLOCATION_INFO

```
# get
<module-index> P4E_ALLOCATION_INFO ?
```

Description

Display information about which PEs that are available for allocation/assignment and which are currently allocated/assigned to this port.

Actions

get

Parameters

`available`: *hex8*, eight hex bytes (64-bit) mask of available PEs

`allocated`: *hex8*, eight hex bytes (64-bit) mask of PEs assigned to this port

Example

```
# get
input:  1/0 P4E_INFO ?
output: 1/0 P4E_INFO 0x00000000000000F8 0x0000000000000008
```

P4E_ASSIGN

```
# set
<module-index> P4E_ASSIGN <mask>

# get
<module-index> P4E_ASSIGN ?
```

Description

Advanced mode only: Assign previously reserved PEs to a port.

Actions

set, get

Parameters

`mask`: *hex8*, eight hex bytes, a bitmask specifying which PEs should be assigned to this port

Example

```
# set
input:  1/0 P4E_ASSIGN 0x0000000040001fff
output: <OK>

# get
input:  1/0 P4E_ASSIGN ?
output: 1/0 P4E_ASSIGN 0x0000000040001fff
```

P4E_AVAILABLE

```
# get
<module-index> P4E_AVAILABLE ?
```

Description

Simple mode only: Report the number of PEs available for allocation.

Actions

get

Parameters

`available_pe_count`: *integer*, total number of PEs that can be allocated to the port - including the PEs already allocated to the port.

Example

```
# get
input:  1/0 P4E_AVAILABLE ?
output: 1/0 P4E_AVAILABLE 4
```

Vulcan Port

This module contains the **Vulcan port commands**.

The Xena L47 test execution engine has seven states: `off`, `prepare`, `prepare_rdy`, `prerun`, `prerun_rdy`, `running` and `stopped`. Traffic is generated in the `prerun` and running states only, and configuration of parameters is only valid in state `off` except for a few runtime options. Port traffic commands can be given with `P4_TRAFFIC` and port state queried by `P4_STATE`.

- `off` - default state. Entered from stopped or prepare on `OFF` command. This is the only state that allows configuration commands. `p_reset` is also considered a configuration command. Upon entering off state, some internal ''house cleaning''' is done. For example: freeing TCP Connections, clearing test specific counters etc.
- `prepare` - this state is entered from state off on `PREPARE` command. Here internal data structures relevant for the test configuration are created.
- `prepare_rdy` - entered automatically after activities in prepare have completed successfully.

- `prepare_fail` - entered automatically from prepare, if an error occurs. An error could for example be failure to load a configured replay file.
- `prerun` - entered from `prepare_ready` on `PRERUN` command. If enabled, this is where ARP and NDP requests are sent.
- `prerun_rdy` - entered automatically after activities in prerun have completed.
- `running` - entered either from `prepare_ready` or `prerun_ready` on `ON` command. This is where TCP connections are established, payload is generated and connections are closed again.
- `stopping` - entered from `running`, `prerun_ready` or `prerun` on `STOP` command. Stops Rx/Tx traffic. In the `stopping` state, post-test data are calculated and captured packets are saved to files.
- `stopped` - entered automatically after activities in `stopping`.

P4_APTITUDES

```
# get
<module-index>/<port-index> P4_APTITUDES ?
```

Description

Returns the ports aptitudes - i.e. what is possible to configure on the port in terms of features and performance.

Current schema of the BSON document:

```
schema = {
    'chassis': {
        'type': 'int32',
        'required': True,
        'enum': ['CHASSIS_TYPE_UNKNOWN',
                 'CHASSIS_TYPE_APPLIANCE',
                 'CHASSIS_TYPE_BAY',
                 'CHASSIS_TYPE_COMPACT',
                 'CHASSIS_TYPE_SAFIRE']
    },
    'tcp_udp': {
        'type': 'document',
        'required': True,
        'properties': {
            'cc': {
                'type': 'int32',
                'required': True,
            },
```

(continues on next page)

(continued from previous page)

```
        }
    },
    'tls': {
        'type': 'document',
        'required': True,
        'properties': {
            'supported': {
                'type': 'bool',
                'required': True,
            },
            'cc': {
                'type': 'int32',
                'required': True,
            }
        }
    }
}
```

Actions

get

Parameters

bson: *short integer list*, the ports aptitudes in BSON format

Example

```
# get
input:  0/1 P4_APTITUDES ?
output: 0/1 P4_APTITUDES 123 123
```

P4_ARP_CONFIG

```
# set
<module-index>/<port-index> P4_ARP_CONFIG <rate> <retrans_timeout>
↪<retries>

# get
<module-index>/<port-index> P4_ARP_CONFIG ?
```

Description

Configure the value of the ARP request transmission rate, retransmission timeout and max. retries.

Actions

set, get

Parameters

`rate`: *integer*, ARP Request transmission rate (requests/sec) - must be larger than 0

`retrans_timeout`: *integer*, ARP Request retransmission timeout [ms] - must be larger than 0

`retries`: *short integer*, maximum ARP Request retransmission retries

Example

```
# set
input:  0/1 P4_ARP_CONFIG 100 100 32
output: <OK>

# get
input:  0/1 P4_ARP_CONFIG ?
output: 0/1 P4_ARP_CONFIG 100 100 32
```

P4_ARP_COUNTERS

```
# get
<module-index>/<port-index> P4_ARP_COUNTERS ?
```

Description

Return total Port ARP protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ARP protocol error statistics since last clear.

`ref_time`: *long integer*, total Port ARP protocol error statistics since last clear.

`invalid_arp_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_request_lookup_failure_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_reply_lookup_failure_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_request_retrans_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_resolved_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_failed_count`: *long integer*, total Port ARP protocol error statistics since last clear.

`arp_table_lookup_failure_count`: *long integer*, total Port ARP protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_ARP_COUNTERS ?
output: 0/1 P4_ARP_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_ARP_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_ARP_RX_COUNTERS ?
```

Description

Return total Port ARP protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ARP protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port ARP protocol receive statistics since last clear.

`arp_request_count`: *long integer*, total Port ARP protocol receive statistics since last clear.

`arp_reply_count`: *long integer*, total Port ARP protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_ARP_RX_COUNTERS ?
output: 0/1 P4_ARP_RX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_ARP_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_ARP_TX_COUNTERS ?
```

Description

Return total Port ARP protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ARP protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port ARP protocol transmit statistics since last clear.

`arp_request_count`: *long integer*, total Port ARP protocol transmit statistics since last clear.

`arp_reply_count`: *long integer*, total Port ARP protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_ARP_TX_COUNTERS ?
output: 0/1 P4_ARP_TX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_CAPABILITIES

```
# get
<module-index>/<port-index> P4_CAPABILITIES ?
```

Description

Report the speeds supported by the L47 port.

Actions

get

Parameters

`auto`: *short integer*, the speeds supported by the L47 port

`N100_mbps`: *short integer*, the speeds supported by the L47 port

`N1_gbps`: *short integer*, the speeds supported by the L47 port

`N2_5_gbps`: *short integer*, the speeds supported by the L47 port

`N5_gbps`: *short integer*, the speeds supported by the L47 port

`N10_gbps`: *short integer*, the speeds supported by the L47 port

`N25_gbps`: *short integer*, the speeds supported by the L47 port

`N40_gbps`: *short integer*, the speeds supported by the L47 port

`N50_gbps`: *short integer*, the speeds supported by the L47 port

`N100_gbps`: *short integer*, the speeds supported by the L47 port

Example

```
# get
input:  0/1 P4_CAPABILITIES ?
output: 0/1 P4_CAPABILITIES 0 1 1 1 1 0 0 0 0
```

P4_CAPTURE

```
# set
<module-index>/<port-index> P4_CAPTURE <on_off>

# get
<module-index>/<port-index> P4_CAPTURE ?
```

Description

Starts or stops packet capture on this port.

Actions

set, get

Parameters

on_off: *byte*, specifying whether to capture traffic on this port

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P4_CAPTURE OFF
output: <OK>

# get
input:  0/1 P4_CAPTURE ?
output: 0/1 P4_CAPTURE OFF
```

P4_CAPTURE_GET_FIRST

```
# get
<module-index>/<port-index> P4_CAPTURE_GET_FIRST ?
```

Description

Returns the first captured frame on the port. Command is only valid when port is in state STOPPED

Actions

get

Parameters

index: *integer*, the first captured frame on the port

second: *integer*, the first captured frame on the port

microsecond: *integer*, the first captured frame on the port

capture_length: *integer*, the first captured frame on the port

frame_length: *integer*, the first captured frame on the port

frame: *hex list*, the first captured frame on the port

Example

```
# get
input:  0/1 P4_CAPTURE_GET_FIRST ?
output: 0/1 P4_CAPTURE_GET_FIRST 0 6325709 706541 60 60␣
↪0xFFFFFFFFFFFF04F40A0002010806....
```

P4_CAPTURE_GET_NEXT

```
# get
<module-index>/<port-index> P4_CAPTURE_GET_NEXT ?
```

Description

Returns the next captured frame on the port. Command is only valid when port is in state STOPPED

Actions

get

Parameters

index: *integer*, the next captured frame on the port

second: *integer*, the next captured frame on the port

microsecond: *integer*, the next captured frame on the port

capture_length: *integer*, the next captured frame on the port

frame_length: *integer*, the next captured frame on the port

`frame`: *hex list*, the next captured frame on the port

Example

```
# get
input:  0/1 P4_CAPTURE_GET_NEXT ?
output: 0/1 P4_CAPTURE_GET_NEXT 1 6325709 706551 42 42␣
↪0x04F40A00020104F40A0F00000806...
```

P4_CLEAR

```
# set
<module-index>/<port-index> P4_CLEAR
```

Description

Set the Port State to OFF and delete all configured Connection Groups for the port.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4_CLEAR
output: <OK>
```

P4_CLEAR_COUNTERS

```
# set
<module-index>/<port-index> P4_CLEAR_COUNTERS
```

Description

Clears all run-time port counters.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4_CLEAR_COUNTERS
output: <OK>
```

P4_DEV_NAME

```
# get
<module-index>/<port-index> P4_DEV_NAME ?
```

Description

Report the name of the device (NIC) on which the port is located.

Actions

get

Parameters

`name`: *string*, the name of the device (NIC) on which the port is located.

Example

```
# get
input:  0/1 P4_DEV_NAME ?
output: 0/1 P4_DEV_NAME "name"
```

P4_ETH_COUNTERS

```
# get
<module-index>/<port-index> P4_ETH_COUNTERS ?
```

Description

Return total port Ethernet statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port Ethernet statistics since last clear.

`ref_time`: *long integer*, total port Ethernet statistics since last clear.

`tx_error_count`: *long integer*, total port Ethernet statistics since last clear.

`rx_error_count`: *long integer*, total port Ethernet statistics since last clear.

`rx_packet_lost_count`: *long integer*, total port Ethernet statistics since last clear.

Example

```
# get
input:  0/1 P4_ETH_COUNTERS ?
output: 0/1 P4_ETH_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_ETH_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_ETH_RX_COUNTERS ?
```

Description

Return total port Ethernet receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port Ethernet receive statistics since last clear.

`ref_time`: *long integer*, total port Ethernet receive statistics since last clear.

`bits_per_sec`: *long integer*, total port Ethernet receive statistics since last clear.

`packets_per_sec`: *long integer*, total port Ethernet receive statistics since last clear.

`byte_count`: *long integer*, total port Ethernet receive statistics since last clear.

`packet_count`: *long integer*, total port Ethernet receive statistics since last clear.

Example

```
# get
input:  0/1 P4_ETH_RX_COUNTERS ?
output: 0/1 P4_ETH_RX_COUNTERS 3620000 3610000 0 0 2173
```

P4_ETH_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_ETH_TX_COUNTERS ?
```

Description

Return total port Ethernet transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port Ethernet transmit statistics since last clear.

`ref_time`: *long integer*, total port Ethernet transmit statistics since last clear.

`bits_per_sec`: *long integer*, total port Ethernet transmit statistics since last clear.

`packets_per_sec`: *long integer*, total port Ethernet transmit statistics since last clear.

`byte_count`: *long integer*, total port Ethernet transmit statistics since last clear.

`packet_count`: *long integer*, total port Ethernet transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_ETH_TX_COUNTERS ?
output: 0/1 P4_ETH_TX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4_FW_VER

```
# get
<module-index>/<port-index> P4_FW_VER ?
```

Description

Report the firmware version of the port (NIC).

Actions

get

Parameters

`major`: *integer*, the firmware version of the port (NIC)

`minor`: *integer*, the firmware version of the port (NIC)

Example

```
# get
input:  0/1 P4_FW_VER ?
output: 0/1 P4_FW_VER 1 1
```

P4_ICMP_COUNTERS

```
# get
<module-index>/<port-index> P4_ICMP_COUNTERS ?
```

Description

Return total Port ICMP protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ICMP protocol error statistics since last clear.

`ref_time`: *long integer*, total Port ICMP protocol error statistics since last clear.

`invalid_icmp_count`: *long integer*, total Port ICMP protocol error statistics since last clear.

`unknown_icmp_count`: *long integer*, total Port ICMP protocol error statistics since last clear.

`invalid_icmpv6_count`: *long integer*, total Port ICMP protocol error statistics since last clear.

`unknown_icmpv6_count`: *long integer*, total Port ICMP protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_ICMP_COUNTERS ?
output: 0/1 P4_ICMP_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4_ICMP_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_ICMP_RX_COUNTERS ?
```

Description

Return total Port ICMP protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`icmp_echo_reqest_count`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`icmp_echo_reply_count`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`icmp_dest_unknown_count`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`icmp_time_excessive_count`: *long integer*, total Port ICMP protocol receive statistics since last clear.

`icmpv6_count`: *long integer*, total Port ICMP protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_ICMP_RX_COUNTERS ?
output: 0/1 P4_ICMP_RX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4_ICMP_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_ICMP_TX_COUNTERS ?
```

Description

Return total Port ICMP protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`icmp_echo_reqest_count`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`icmp_echo_reply_count`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`icmp_dest_unknown_count`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`icmp_time_excessive_count`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

`icmpv6_count`: *long integer*, total Port ICMP protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_ICMP_TX_COUNTERS ?
output: 0/1 P4_ICMP_TX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4_IPV4_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV4_COUNTERS ?
```

Description

Return total Port IPv4 protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv4 protocol error statistics since last clear.

`ref_time`: *long integer*, total Port IPv4 protocol error statistics since last clear.

`checksum_error_count`: *long integer*, total Port IPv4 protocol error statistics since last clear.

`invalid_packet_count`: *long integer*, total Port IPv4 protocol error statistics since last clear.

`unknown_packet_count`: *long integer*, total Port IPv4 protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV4_COUNTERS ?
output: 0/1 P4_IPV4_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_IPV4_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV4_RX_COUNTERS ?
```

Description

Return total Port IPv4 protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv4 protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port IPv4 protocol receive statistics since last clear.

`packet_count`: *long integer*, total Port IPv4 protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV4_RX_COUNTERS ?
output: 0/1 P4_IPV4_RX_COUNTERS 123456789123 123456789123 123456789123
```

P4_IPV4_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV4_TX_COUNTERS ?
```

Description

Return total Port IPv4 protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv4 protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port IPv4 protocol transmit statistics since last clear.

`packet_count`: *long integer*, total Port IPv4 protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV4_TX_COUNTERS ?
output: 0/1 P4_IPV4_TX_COUNTERS 123456789123 123456789123 123456789123
```

P4_IPV6_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV6_COUNTERS ?
```

Description

Return total Port IPv6 protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv6 protocol error statistics since last clear.

`ref_time`: *long integer*, total Port IPv6 protocol error statistics since last clear.

`invalid_packet_count`: *long integer*, total Port IPv6 protocol error statistics since last clear.

`unknown_packet_count`: *long integer*, total Port IPv6 protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV6_COUNTERS ?
output: 0/1 P4_IPV6_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_IPV6_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV6_RX_COUNTERS ?
```

Description

Return total Port IPv6 protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv6 protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port IPv6 protocol receive statistics since last clear.

`packet_count`: *long integer*, total Port IPv6 protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV6_RX_COUNTERS ?
output: 0/1 P4_IPV6_RX_COUNTERS 123456789123 123456789123 123456789123
```

P4_IPV6_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_IPV6_TX_COUNTERS ?
```

Description

Return total Port IPv6 protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port IPv6 protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port IPv6 protocol transmit statistics since last clear.

`packet_count`: *long integer*, total Port IPv6 protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_IPV6_TX_COUNTERS ?
output: 0/1 P4_IPV6_TX_COUNTERS 123456789123 123456789123 123456789123
```

P4_LICENSE_INFO

```
# get
<module-index>/<port-index> P4_LICENSE_INFO ?
```

Description

Returns the information on the license assigned to the port - if any.

Actions

get

Parameters

present: *byte*, **the information on the license assigned to the port**

- NOT_PRESENT = 0
- PRESENT = 1

speed: *byte*, **the information on the license assigned to the port**

- UNDEFINED = 0
- F1G = 1
- F2_5G = 2
- F5G = 3
- F10G = 4
- F25G = 5
- F40G = 6
- F50G = 7
- F100G = 8

permanency: *byte*, **the information on the license assigned to the port**

- NON_PERMANENT = 0
- PERMANENT = 1

expiration: *long integer*, the information on the license assigned to the port

Example

```
# get
input:  0/1 P4_LICENSE_INFO ?
output: 0/1 P4_LICENSE_INFO NOT_PRESENT UNDEFINED NON_PERMANENT␣
↪123456789123
```

P4_MAX_PACKET_RATE

```
# set
<module-index>/<port-index> P4_MAX_PACKET_RATE <mode> <rate> <time_
↪window>

# get
<module-index>/<port-index> P4_MAX_PACKET_RATE ?
```

Description

Specifies the maximum number of packets per second allowed to be transmitted on the port.

Actions

set, get

Parameters

mode: *byte*, **specifies the mode of the max. pps mechanism**

- AUTOMATIC = 0
- MANUAL = 1

rate: *integer*, maximum number of packets per second to transmit on this port

time_window: *integer*, time window [us] to measure the pps rate

Example

```
# set
input:  0/1 P4_MAX_PACKET_RATE AUTOMATIC 1 1
output: <OK>

# get
input:  0/1 P4_MAX_PACKET_RATE ?
output: 0/1 P4_MAX_PACKET_RATE AUTOMATIC 1 1
```

P4_NDP_CONFIG

```
# set
<module-index>/<port-index> P4_NDP_CONFIG <rate> <retrans_timeout>
↪<retries>

# get
<module-index>/<port-index> P4_NDP_CONFIG ?
```

Description

Configure the value of the NDP Neighbor Solicitation transmission rate, retransmission timeout and max. retries.

Actions

set, get

Parameters

`rate`: *integer*, NDP Neighbor Solicitation transmission rate (requests/sec) - must be larger than 0

`retrans_timeout`: *integer*, NDP Neighbor Solicitation retransmission timeout [ms] - must be larger than 0

`retries`: *short integer*, maximum NDP Neighbor Solicitation retransmission retries

Example

```
# set
input:  0/1 P4_NDP_CONFIG 100 100 32
output: <OK>

# get
input:  0/1 P4_NDP_CONFIG ?
output: 0/1 P4_NDP_CONFIG 100 100 32
```

P4_NDP_COUNTERS

```
# get
<module-index>/<port-index> P4_NDP_COUNTERS ?
```

Description

Return total Port NDP protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port NDP protocol error statistics since last clear.

`ref_time`: *long integer*, total Port NDP protocol error statistics since last clear.

`invalid_ndp_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_request_lookup_failure_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_reply_lookup_failure_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_request_retrans_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_resolved_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_failed_count`: *long integer*, total Port NDP protocol error statistics since last clear.

`ndp_table_lookup_failure_count`: *long integer*, total Port NDP protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_NDP_COUNTERS ?
output: 0/1 P4_NDP_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_NDP_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_NDP_RX_COUNTERS ?
```

Description

Return total Port NDP protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port NDP protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port NDP protocol receive statistics since last clear.

`ndp_request_count`: *long integer*, total Port NDP protocol receive statistics since last clear.

`ndp_reply_count`: *long integer*, total Port NDP protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_NDP_RX_COUNTERS ?
output: 0/1 P4_NDP_RX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_NDP_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_NDP_TX_COUNTERS ?
```

Description

Return total Port NDP protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port NDP protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port NDP protocol transmit statistics since last clear.

`ndp_request_count`: *long integer*, total Port NDP protocol transmit statistics since last clear.

`ndp_reply_count`: *long integer*, total Port NDP protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_NDP_TX_COUNTERS ?
output: 0/1 P4_NDP_TX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123
```

P4_PCI_INFO

```
# get
<module-index>/<port-index> P4_PCI_INFO ?
```

Description

Report the port PCI info.

Actions

get

Parameters

`vendor_id`: *hex*, the port PCI info

`device_id`: *hex*, the port PCI info

`sub_vendor_id`: *hex*, the port PCI info

`sub_device_id`: *hex*, the port PCI info

`rev`: *integer*, the port PCI info

Example

```
# get
input:  0/1 P4_PCI_INFO ?
output: 0/1 P4_PCI_INFO 0x57 0x57 0x57 0x57 1
```

P4_PORT_COUNTERS

```
# get
<module-index>/<port-index> P4_PORT_COUNTERS ?
```

Description

Return total port transmit error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port transmit error statistics since last clear.

`ref_time`: *long integer*, total port transmit error statistics since last clear.

`invalid_eth_count`: *long integer*, total port transmit error statistics since last clear.

`unknown_eth_count`: *long integer*, total port transmit error statistics since last clear.

`mismatch_vlan_error_count`: *long integer*, total port transmit error statistics since last clear.

`pkt_rate_limit_count`: *long integer*, total port transmit error statistics since last clear.

Example

```
# get
input:  0/1 P4_PORT_COUNTERS ?
output: 0/1 P4_PORT_COUNTERS 3620000 3610000 0 0 2173
```

P4_PORT_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_PORT_RX_COUNTERS ?
```

Description

Return total port receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port receive statistics since last clear.

`ref_time`: *long integer*, total port receive statistics since last clear.

`vlan_packet_count`: *long integer*, total port receive statistics since last clear.

`bits_per_sec`: *long integer*, total port receive statistics since last clear.

`byte_count`: *long integer*, total port receive statistics since last clear.

Example

```
# get
input:  0/1 P4_PORT_RX_COUNTERS ?
output: 0/1 P4_PORT_RX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_PORT_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_PORT_TX_COUNTERS ?
```

Description

Return total port transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total port transmit statistics since last clear.

`ref_time`: *long integer*, total port transmit statistics since last clear.

`vlan_packet_count`: *long integer*, total port transmit statistics since last clear.

`bits_per_sec`: *long integer*, total port transmit statistics since last clear.

`byte_count`: *long integer*, total port transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_PORT_TX_COUNTERS ?
output: 0/1 P4_PORT_TX_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_PORT_TYPE

```
# get
<module-index>/<port-index> P4_PORT_TYPE ?
```

Description

Report the port type. The different possible ports are divided into types.

Actions

get

Parameters

`type_number`: *integer*, the L47 port type

`type_string`: *string*, the L47 port type

Example

```
# get
input:  0/1 P4_PORT_TYPE ?
output: 0/1 P4_PORT_TYPE 1 "type"
```

P4_RX_MTU

```
# get
<module-index>/<port-index> P4_RX_MTU ?
```

Description

Return histogram over received (layer 3) packets sizes in 1 byte intervals. Each bin represents a packet size in the interval [576..1500] bytes.

Actions

get

Parameters

`bins`: *short integer list*, histogram over received (layer 3) packets sizes in 1 byte intervals.

Example

```
# get
input:  0/1 P4_RX_MTU ?
output: 0/1 P4_RX_MTU 123 123
```

P4_RX_PACKET_SIZE

```
# get
<module-index>/<port-index> P4_RX_PACKET_SIZE ?
```

Description

Return a histogram over received (layer 2) packets sizes in 100 bytes intervals.

Actions

get

Parameters

`current_time`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`ref_time`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`bin_00`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`bin_01`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`bin_02`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`bin_03`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

`bin_04`: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_05: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_06: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_07: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_08: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_09: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_10: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_11: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_12: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_13: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_14: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

bin_15: *long integer*, a histogram over received (layer 2) packets sizes in 100 bytes intervals.

Example

```
# get
input:  0/1 P4_RX_PACKET_SIZE ?
output: 0/1 P4_RX_PACKET_SIZE 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123
```

P4_SPEEDSELECTION

```
# set
<module-index>/<port-index> P4_SPEEDSELECTION <speed>

# get
<module-index>/<port-index> P4_SPEEDSELECTION ?
```

Description

Sets the port speed. The selected speed must be one of the speeds supported by the port, which can be retrieved with P4_CAPABILITIES.

Actions

set, get

Parameters

speed: *byte*, **specifies the speed mode of the port**

- AUTO = 0
- F100M = 1
- F1G = 2
- F2_5G = 3
- F5G = 4
- F10G = 5
- F25G = 6
- F40G = 7
- F50G = 8
- F100G = 9

Example

```
# set
input:  0/1 P4_SPEEDSELECTION AUTO
output: <OK>

# get
input:  0/1 P4_SPEEDSELECTION ?
output: 0/1 P4_SPEEDSELECTION AUTO
```

P4_STATE

```
# get
<module-index>/<port-index> P4_STATE ?
```

Description

Display the current state of the L47 port.

Actions

get

Parameters

state: *byte*, **the current state of the L47 port**

- OFF = 0
- PREPARE = 1
- PREPARE_RDY = 2
- PREPARE_FAIL = 3
- PRERUN = 4
- PRERUN_RDY = 5
- RUNNING = 6
- STOPPING = 7
- STOPPED = 8

Example

```
# get
input:  0/1 P4_STATE ?
output: 0/1 P4_STATE OFF
```

P4_STATE_STATUS

```
# get
<module-index>/<port-index> P4_STATE_STATUS ?
```

Description

Returns status of the last port state change. If the port state has changed to PREPARE_FAIL, the status contains information about the reason for the fail. Currently the status will be "OK" in all other states.

Actions

get

Parameters

`status`: *string*, status of the last port state change

Example

```
# get
input:  0/1 P4_STATE_STATUS ?
output: 0/1 P4_STATE_STATUS "OK"
```

P4_TCP_COUNTERS

```
# get
<module-index>/<port-index> P4_TCP_COUNTERS ?
```

Description

Return total Port TCP protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port TCP protocol error statistics since last clear.

`ref_time`: *long integer*, total Port TCP protocol error statistics since last clear.

`checksum_error_count`: *long integer*, total Port TCP protocol error statistics since last clear.

`invalid_tcp_count`: *long integer*, total Port TCP protocol error statistics since last clear.

`tcp_lookup_failure_count`: *long integer*, total Port TCP protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_TCP_COUNTERS ?
output: 0/1 P4_TCP_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_TCP_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_TCP_RX_COUNTERS ?
```

Description

Return total Port TCP protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port TCP protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port TCP protocol receive statistics since last clear.

`packet_count`: *long integer*, total Port TCP protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_TCP_RX_COUNTERS ?
output: 0/1 P4_TCP_RX_COUNTERS 123456789123 123456789123 123456789123
```

P4_TCP_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_TCP_TX_COUNTERS ?
```

Description

Return total Port TCP protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port TCP protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port TCP protocol transmit statistics since last clear.

`packet_count`: *long integer*, total Port TCP protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_TCP_TX_COUNTERS ?
output: 0/1 P4_TCP_TX_COUNTERS 123456789123 123456789123 123456789123
```

P4_TRAFFIC

```
# set
<module-index>/<port-index> P4_TRAFFIC <traffic_state>
```

Description

Gives a traffic state command to a L47 port.

Actions

set

Parameters

traffic_state: *byte*, the traffic state command issued to the port

- OFF = 0
- ON = 1
- STOP = 2
- PREPARE = 3
- PRERUN = 4

Example

```
# set
input:  0/1 P4_TRAFFIC OFF
output: <OK>
```

P4_TX_MTU

```
# get
<module-index>/<port-index> P4_TX_MTU ?
```

Description

Return histogram over transmitted (layer 3) packets sizes in 1 byte intervals. Each bin represents a packet size in the interval [576..1500] bytes.

Actions

get

Parameters

`bins`: *short integer list*, histogram over transmitted (layer 3) packets sizes in 1 byte intervals.

Example

```
# get
input:  0/1 P4_TX_MTU ?
output: 0/1 P4_TX_MTU 123 123
```

P4_TX_PACKET_SIZE

```
# get
<module-index>/<port-index> P4_TX_PACKET_SIZE ?
```

Description

Return histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

Actions

get

Parameters

`current_time`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`ref_time`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_00`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_01`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_02`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_03`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_04`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_05`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_06`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_07`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_08`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_09`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_10`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_11`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_12`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_13`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_14`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

`bin_15`: *long integer*, histogram over transmitted (layer 2) packets sizes in 100 bytes intervals.

Example

```
# get
input:  0/1 P4_TX_PACKET_SIZE ?
output: 0/1 P4_TX_PACKET_SIZE 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123
```

P4_UDP_COUNTERS

```
# get
<module-index>/<port-index> P4_UDP_COUNTERS ?
```

Description

Return total Port UDP protocol error statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port UDP protocol error statistics since last clear.

`ref_time`: *long integer*, total Port UDP protocol error statistics since last clear.

`checksum_error_count`: *long integer*, total Port UDP protocol error statistics since last clear.

`invalid_udp_count`: *long integer*, total Port UDP protocol error statistics since last clear.

`udp_lookup_failure_count`: *long integer*, total Port UDP protocol error statistics since last clear.

Example

```
# get
input:  0/1 P4_UDP_COUNTERS ?
output: 0/1 P4_UDP_COUNTERS 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4_UDP_RX_COUNTERS

```
# get
<module-index>/<port-index> P4_UDP_RX_COUNTERS ?
```

Description

Return total Port UDP protocol receive statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port UDP protocol receive statistics since last clear.

`ref_time`: *long integer*, total Port UDP protocol receive statistics since last clear.

`packet_count`: *long integer*, total Port UDP protocol receive statistics since last clear.

Example

```
# get
input:  0/1 P4_UDP_RX_COUNTERS ?
output: 0/1 P4_UDP_RX_COUNTERS 123456789123 123456789123 123456789123
```

P4_UDP_TX_COUNTERS

```
# get
<module-index>/<port-index> P4_UDP_TX_COUNTERS ?
```

Description

Return total Port UDP protocol transmit statistics since last clear.

Actions

get

Parameters

`current_time`: *long integer*, total Port UDP protocol transmit statistics since last clear.

`ref_time`: *long integer*, total Port UDP protocol transmit statistics since last clear.

`packet_count`: *long integer*, total Port UDP protocol transmit statistics since last clear.

Example

```
# get
input:  0/1 P4_UDP_TX_COUNTERS ?
output: 0/1 P4_UDP_TX_COUNTERS 123456789123 123456789123 123456789123
```

P4_VLAN_OFFLOAD

```
# set
<module-index>/<port-index> P4_VLAN_OFFLOAD <offload>

# get
<module-index>/<port-index> P4_VLAN_OFFLOAD ?
```

Description

Specifies if 802.1Q VLAN tag should be inserted and stripped by the Ethernet device. If VLAN Offload is switched ON, VLAN tags will not be present in frames captured by the L47 Server.

Actions

set, get

Parameters

`offload`: *byte*, specifies if VLAN Offload is enabled

- `OFF = 0`
- `ON = 1`

Example

```
# set
input:  0/1 P4_VLAN_OFFLOAD OFF
output: <OK>

# get
input:  0/1 P4_VLAN_OFFLOAD ?
output: 0/1 P4_VLAN_OFFLOAD OFF
```

Vulcan Connection Group

This module contains the **Vulcan connection group commands** that deal with configuration of TCP connections and are specific to Vulcan. The commands have the form `P4G_<xxx>` and require a module index id and a port index id, and a connection group index id.

A *Connection Group (CG)* is the basic building block when creating stateful traffic. A CG consists of a number of TCP connections - between one and millions. The CG has a role, which is either client or server. In order to create TCP connections between two ports on a Vulcan chassis, two matching CGs must be configured - one on each port - one configured as client and the other configured as server. The number of connections in a CG, is defined by the server range and the client range.

A server/client range is a number of TCP connection endpoints defined by a number of IP addresses and a number of TCP ports. A server/client range is configured by specifying a start IP address, a number of IP addresses, a start TCP port and a number of TCP addresses. The number of clients is the number of client IP addresses times the number of client TCP ports, and the same goes for the number of servers. The number of TCP connections in a CG is the number of clients times the number of servers, that is TCP connections are created from all clients in the CG to all servers in the CG.

Example:

A CG containing 100 clients on port 0 and 10 servers on port 1 can be configured the following way:

```
1/0 P4G_CREATE [0]
1/1 P4G_CREATE [0]
1/0 P4G_ROLE [0] CLIENT
1/1 P4G_ROLE [0] SERVER

1/0 P4G_CLIENT_RANGE [0] 10.0.1.1 10 5000 10
1/0 P4G_SERVER_RANGE [0] 10.0.2.1 10 80 1
1/1 P4G_CLIENT_RANGE [0] 10.0.1.1 10 5000 10
1/1 P4G_SERVER_RANGE [0] 10.0.2.1 10 80 1
```

Important: CG index must start from 0.

Now Port 0 contains 100 clients - 10 different TCP ports on 10 different IP addresses, and Port 1 contains 10 servers - 1 TCP port on 10 different IP addresses. When starting traffic on Port 0 and 1, 1000 TCP connections will be established - from all clients to all servers.

Note: When configuring a CG, both client **AND** server range must be configured on both CGs - that is, the server CG must also know the client range and vice versa.

The CG must be configured with a *Load Profile*, which is an envelope over the TCP connection's lifetime. A connection in the CG goes through three phases. The Load Profile defines a start time and a duration of each of these phases. During the ramp-up phase connections are established at a rate defined by the number of connections divided by the ramp-up duration. During the steady-state phase connections may transmit and receive payload data, depending on the configuration of test application and test scenario for the CG. During the ramp-down phase connections are closed at a rate defined by the number of connections divided by the ramp-up duration, if they were not already closed as a result of the traffic scenario configured.

Example:

The 1000 connections configured above will be ramped up in 1 second - starting immediately - will live for 10 seconds, and will be ramped down in 2 seconds with the following configuration:

```
1/0 P4G_LP_TIME_SCALE [0] SECONDS
1/1 P4G_LP_TIME_SCALE [0] SECONDS

1/0 P4G_LP_SHAPE [0] 0 1 10 2
1/1 P4G_LP_SHAPE [0] 0 1 10 2
```

Note: Just like client and server range, both the client and server CG must be configured with the Load Profile.

Next the CG must be configured with a test application, which defines what type of traffic is transported in the TCP payload. Currently there are two kinds of test applications:

- `NONE` means that no payload is sent on the TCP connections. This test application is suitable for a test, where the only purpose is to measure TCP connection open and close rates.
- `RAW` means that the TCP connections transmit and receive user defined raw data. The contents of the raw TCP payload can be configured using the `P4G_RAW_PAYLOAD` command. Raw TCP payload can also be specified as random and incrementing data.

Using test application `RAW`, the CG must also be configured with a test scenario, which defines the data flow between the TCP client and server. Currently the following test scenarios can be configured: `DOWNLOAD`, `UPLOAD`, `BOTH` or `ECHO`.

Example:

The CG defined above is configured to transmit random payload data from the servers to the clients after the clients have transmitted (and the servers received) a download request, with the

following commands:

```
1/0 P4G_TEST_APPLICATION [0] RAW
1/1 P4G_TEST_APPLICATION [0] RAW

1/0 P4G_RAW_TEST_SCENARIO [0] DOWNLOAD
1/1 P4G_RAW_TEST_SCENARIO [0] DOWNLOAD

1/0 P4G_RAW_HAS_DOWNLOAD_REQ [0] YES
1/1 P4G_RAW_HAS_DOWNLOAD_REQ [0] YES

1/0 P4G_RAW_PAYLOAD_TYPE [0] RANDOM
1/1 P4G_RAW_PAYLOAD_TYPE [0] RANDOM
```

By combining several CGs on a port, it is possible to create more complex traffic scenarios and more complex Load Profile shapes than the individual CG allows.

P4G_APP_REPLAY_COUNTERS

```
# get
<module-index>/<port-index> P4G_APP_REPLAY_COUNTERS [<group_index>] ?
```

Description

Returns NAT collisions of a replay application.

Actions

get

Parameters

`current_time`: *long integer*, NAT collisions of a replay application.

`ref_time`: *long integer*, NAT collisions of a replay application.

`nat_collision_count`: *long integer*, NAT collisions of a replay application.

Example

```
# get
input:  0/1 P4G_APP_REPLAY_COUNTERS [0] ?
output: 0/1 P4G_APP_REPLAY_COUNTERS [0] 123456789123 123456789123␣
↪123456789123
```

P4G_APP_TRANSACTION_COUNTERS

```
# get
<module-index>/<port-index> P4G_APP_TRANSACTION_COUNTERS [<group_index>
↪] ?
```

Description

Returns request/response transaction statistics.

Actions

get

Parameters

`current_time`: *long integer*, request/response transaction statistics.

`ref_time`: *long integer*, request/response transaction statistics.

`transaction_count`: *long integer*, request/response transaction statistics.

`transaction_per_second`: *long integer*, request/response transaction statistics.

Example

```
# get
input:  0/1 P4G_APP_TRANSACTION_COUNTERS [0] ?
output: 0/1 P4G_APP_TRANSACTION_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_APP_TRANSACTION_HIST

```
# get
<module-index>/<port-index> P4G_APP_TRANSACTION_HIST [<group_index>] ?
```

Description

Returns a histogram over completed request/response transactions per connection,
with start and interval values as configured by *P4G_TRANSACTION_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over completed request/response transactions per connection

`min_transaction_per_con`: *long integer*, a histogram over completed request/response transactions per connection

`max_transaction_per_con`: *long integer*, a histogram over completed request/response transactions per connection

`avg_transaction_per_con`: *long integer*, a histogram over completed request/response transactions per connection

`start`: *long integer*, a histogram over completed request/response transactions per connection

`interval`: *long integer*, a histogram over completed request/response transactions per connection

`bin_00`: *integer*, a histogram over completed request/response transactions per connection

`bin_01`: *integer*, a histogram over completed request/response transactions per connection

`bin_02`: *integer*, a histogram over completed request/response transactions per connection

`bin_03`: *integer*, a histogram over completed request/response transactions per connection

`bin_04`: *integer*, a histogram over completed request/response transactions per connection

`bin_05`: *integer*, a histogram over completed request/response transactions per connection

`bin_06`: *integer*, a histogram over completed request/response transactions per connection

`bin_07`: *integer*, a histogram over completed request/response transactions per connection

`bin_08`: *integer*, a histogram over completed request/response transactions per connection

`bin_09`: *integer*, a histogram over completed request/response transactions per connection

bin_10: *integer*, a histogram over completed request/response transactions per connection

bin_11: *integer*, a histogram over completed request/response transactions per connection

bin_12: *integer*, a histogram over completed request/response transactions per connection

bin_13: *integer*, a histogram over completed request/response transactions per connection

bin_14: *integer*, a histogram over completed request/response transactions per connection

bin_15: *integer*, a histogram over completed request/response transactions per connection

bin_16: *integer*, a histogram over completed request/response transactions per connection

bin_17: *integer*, a histogram over completed request/response transactions per connection

bin_18: *integer*, a histogram over completed request/response transactions per connection

bin_19: *integer*, a histogram over completed request/response transactions per connection

bin_20: *integer*, a histogram over completed request/response transactions per connection

bin_21: *integer*, a histogram over completed request/response transactions per connection

bin_22: *integer*, a histogram over completed request/response transactions per connection

bin_23: *integer*, a histogram over completed request/response transactions per connection

bin_24: *integer*, a histogram over completed request/response transactions per connection

bin_25: *integer*, a histogram over completed request/response transactions per connection

bin_26: *integer*, a histogram over completed request/response transactions per connection

bin_27: *integer*, a histogram over completed request/response transactions per connection

bin_28: *integer*, a histogram over completed request/response transactions per connection

bin_29: *integer*, a histogram over completed request/response transactions per connection

bin_30: *integer*, a histogram over completed request/response transactions per connection

bin_31: *integer*, a histogram over completed request/response transactions per connection

Example

```
# get
input:  0/1 P4G_APP_TRANSACTION_HIST [0] ?
output: 0/1 P4G_APP_TRANSACTION_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_CLEAR_COUNTERS

```
# set
<module-index>/<port-index> P4G_CLEAR_COUNTERS [<group_index>]
```

Description

Clears all run-time statistics for the CG.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_CLEAR_COUNTERS [0]
output: <OK>
```

P4G_CLEAR_POST_STAT

```
# set
<module-index>/<port-index> P4G_CLEAR_POST_STAT [<group_index>]
```

Description

Clears all TCP CG post-test statistics.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_CLEAR_POST_STAT [0]
output: <OK>
```

P4G_CLIENT_RANGE

```
# set
<module-index>/<port-index> P4G_CLIENT_RANGE [<group_index>] <ipv4_
↪address> <address_count> <start_port> <port_count> <max_address_
↪count>

# get
<module-index>/<port-index> P4G_CLIENT_RANGE [<group_index>] ?
```

Description

Specifies a number of client sockets (ip address, port number)

Actions

set, get

Parameters

`ipv4_address`: *address*, the start IP address of the address range

`address_count`: *integer*, the number of IP addresses

`start_port`: *integer*, the starting port number of the port range

`port_count`: *integer*, the number of ports

`max_address_count`: *integer*, the maximum number of IP addresses that this CG will use, when connection incarnation is set to `REINCARNATE`

Example

```
# set
input:  0/1 P4G_CLIENT_RANGE [0] 192.168.1.100 1 1 1 1
output: <OK>

# get
input:  0/1 P4G_CLIENT_RANGE [0] ?
output: 0/1 P4G_CLIENT_RANGE [0] 192.168.1.100 1 1 1 1
```

P4G_COMMENT

```
# set
<module-index>/<port-index> P4G_COMMENT [<group_index>] <comment>

# get
<module-index>/<port-index> P4G_COMMENT [<group_index>] ?
```

Description

The description of a CG.

Actions

set, get

Parameters

`comment`: *string*, the description of a CG.

Example

```
# set
input:  0/1 P4G_COMMENT [0] "this is a comment"
output: <OK>

# get
input:  0/1 P4G_COMMENT [0] ?
output: 0/1 P4G_COMMENT [0] "this is a comment"
```

P4G_CREATE

```
# set
<module-index>/<port-index> P4G_CREATE [<group_index>]
```

Description

Creates an empty CG with the specified sub-index value.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_CREATE [0]
output: <OK>
```

P4G_DELETE

```
# set
<module-index>/<port-index> P4G_DELETE [<group_index>]
```

Description

Deletes a CG with the specified sub-index value.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_DELETE [0]
output: <OK>
```

P4G_ENABLE

```
# set
<module-index>/<port-index> P4G_ENABLE [<group_index>] <status>

# get
<module-index>/<port-index> P4G_ENABLE [<group_index>] ?
```

Description

Enable/disable/suppress a previously created CG with the specified sub-index value.

Actions

set, get

Parameters

status: OnOffWithSuppress, specifies the state of the CG.

- OFF = 0
- ON = 1
- SUPPRESS = 2

Example

```
# set
input:  0/1 P4G_ENABLE [0] OFF
output: <OK>

# get
```

(continues on next page)

(continued from previous page)

```
input:  0/1 P4G_ENABLE [0] ?
output: 0/1 P4G_ENABLE [0] OFF
```

P4G_INDICES

```
# set
<module-index>/<port-index> P4G_INDICES <group_identifiers>

# get
<module-index>/<port-index> P4G_INDICES ?
```

Description

The full list of CGs on this port. These are the sub-index that are used for the parameters that specify TCP connection behavior.

Actions

set, get

Parameters

`group_identifiers`: *integer list*, list of indices identifying CGs.

Example

```
# set
input:  0/1 P4G_INDICES 0 1
output: <OK>

# get
input:  0/1 P4G_INDICES ?
output: 0/1 P4G_INDICES 0 1
```

P4G_IP_DS_MASK

```
# set
<module-index>/<port-index> P4G_IP_DS_MASK [<group_index>] <ds_mask>

# get
<module-index>/<port-index> P4G_IP_DS_MASK [<group_index>] ?
```

Description

Specify a bit mask to be applied to the DS field. If the fixed value is fixed, the current (calculated) value is curr, and the mask is mask, then the effective DS will be calculated as follows: (fixed AND (NOT mask)) OR (curr AND mask) or in C syntax (fixed & (~mask)) | (curr & mask)

Actions

set, get

Parameters

ds_mask: *hex*, the DS mask to be used.

Example

```
# set
input:  0/1 P4G_IP_DS_MASK [0] 0x0F
output: <OK>

# get
input:  0/1 P4G_IP_DS_MASK [0] ?
output: 0/1 P4G_IP_DS_MASK [0] 0x0F
```

P4G_IP_DS_MINMAX

```
# set
<module-index>/<port-index> P4G_IP_DS_MINMAX [<group_index>] <ds_min>
↪<ds_max>

# get
<module-index>/<port-index> P4G_IP_DS_MINMAX [<group_index>] ?
```

Description

Configure the min and max values of the range for the calculated part of the DS value. Both values are included in the range. Relevant when `P4G_IP_DS_TYPE` is set to `INCREMENT` or `RANDOM`.

Actions

set, get

Parameters

`ds_min`: *hex*, minimum value for the calculated part of DS

`ds_max`: *hex*, maximum value for the calculated part of DS

Example

```
# set
input:  0/1 P4G_IP_DS_MINMAX [0] 0x00 0x0F
output: <OK>

# get
input:  0/1 P4G_IP_DS_MINMAX [0] ?
output: 0/1 P4G_IP_DS_MINMAX [0] 0x00 0x0F
```

P4G_IP_DS_STEP

```
# set
<module-index>/<port-index> P4G_IP_DS_STEP [<group_index>] <ds_step>

# get
<module-index>/<port-index> P4G_IP_DS_STEP [<group_index>] ?
```

Description

Specifies the incrementing step size for the calculated part of the DS value. Relevant when P4G_IP_DS_TYPE is set to INCREMENT.

Actions

set, get

Parameters

ds_step: *hex*, the incrementing step size for DS.

Example

```
# set
input:  0/1 P4G_IP_DS_STEP [0] 0x01
output: <OK>

# get
input:  0/1 P4G_IP_DS_STEP [0] ?
output: 0/1 P4G_IP_DS_STEP [0] 0x01
```

P4G_IP_DS_TYPE

```
# set
<module-index>/<port-index> P4G_IP_DS_TYPE [<group_index>] <ds_type>

# get
<module-index>/<port-index> P4G_IP_DS_TYPE [<group_index>] ?
```

Description

Configure the mode of the DS field of the IP header of this CG.

Actions

set, get

Parameters

ds_type: *byte*, **specifying how to fill out the DS field**

- FIXED = 0
- INCREMENT = 1
- RANDOM = 2

Example

```
# set
input:  0/1 P4G_IP_DS_TYPE [0] FIXED
output: <OK>

# get
input:  0/1 P4G_IP_DS_TYPE [0] ?
output: 0/1 P4G_IP_DS_TYPE [0] FIXED
```

P4G_IP_DS_VALUE

```
# set
<module-index>/<port-index> P4G_IP_DS_VALUE [<group_index>] <ds_value>

# get
<module-index>/<port-index> P4G_IP_DS_VALUE [<group_index>] ?
```

Description

Specify the (FIXED) value used for DS.

Actions

set, get

Parameters

`ds_value`: *hex*, the fixed DS value to be used

Example

```
# set
input:  0/1 P4G_IP_DS_VALUE [0] 0x00
output: <OK>

# get
input:  0/1 P4G_IP_DS_VALUE [0] ?
output: 0/1 P4G_IP_DS_VALUE [0] 0x00
```

P4G_IP_VERSION

```
# set
<module-index>/<port-index> P4G_IP_VERSION [<group_index>] <version_
↪number>

# get
<module-index>/<port-index> P4G_IP_VERSION [<group_index>] ?
```

Description

Specifies either IPv4 or IPv6.

Actions

set, get

Parameters

version_number: `VulcanIPVersion`, **IP version**

- `IPV4 = 4`
- `IPV6 = 6`

Example

```
# set
input:  0/1 P4G_IP_VERSION [0] IPV4
output: <OK>

# get
input:  0/1 P4G_IP_VERSION [0] ?
output: 0/1 P4G_IP_VERSION [0] IPV4
```

P4G_IPV6_CLIENT_RANGE

```
# set
<module-index>/<port-index> P4G_IPV6_CLIENT_RANGE [<group_index>]
 ↪<ipv6_address> <address_count> <start_port> <port_count> <max_
 ↪address_count>

# get
<module-index>/<port-index> P4G_IPV6_CLIENT_RANGE [<group_index>] ?
```

Description

Specifies the number of client sockets (IPv6 address, port number).

Actions

set, get

Parameters

`ipv6_address`: *string*, the start ip address of the address range

`address_count`: *integer*, the number of IPv6 addresses

`start_port`: *integer*, the start port number of the port range

`port_count`: *integer*, the number of ports

`max_address_count`: *long integer*, the maximum number of IPv6 addresses that this CG will use, when connection incarnation is set to `REINCARNATE`

Example

```
# set
input:  0/1 P4G_IPV6_CLIENT_RANGE [0] ::1 1 1 1 1
output: <OK>

# get
input:  0/1 P4G_IPV6_CLIENT_RANGE [0] ?
output: 0/1 P4G_IPV6_CLIENT_RANGE [0] ::1 1 1 1 1
```

P4G_IPV6_FLOW_LABEL

```
# set
<module-index>/<port-index> P4G_IPV6_FLOW_LABEL [<group_index>] <flow_
↪label>

# get
<module-index>/<port-index> P4G_IPV6_FLOW_LABEL [<group_index>] ?
```

Description

Configure the value of the flow label field of the IPv6 header.

Actions

set, get

Parameters

`flow_label`: *hex2*, value of the traffic class field (only lowest 20 bits are valid)

Example

```
# set
input:  0/1 P4G_IPV6_FLOW_LABEL [0] 0x0000
output: <OK>

# get
input:  0/1 P4G_IPV6_FLOW_LABEL [0] ?
output: 0/1 P4G_IPV6_FLOW_LABEL [0] 0x0000
```

P4G_IPV6_SERVER_RANGE

```
# set
<module-index>/<port-index> P4G_IPV6_SERVER_RANGE [<group_index>]
 ↪<ipv6_address> <address_count> <start_port> <port_count>

# get
<module-index>/<port-index> P4G_IPV6_SERVER_RANGE [<group_index>] ?
```

Description

Specifies the number of server sockets (IPv6 address, port number)

Actions

set, get

Parameters

`ipv6_address`: *string*, the start IPv6 address of the address range

`address_count`: *integer*, the number of IPv6 addresses

`start_port`: *integer*, the start port number of the port range

`port_count`: *integer*, the number of ports

Example

```
# set
input:  0/1 P4G_IPV6_SERVER_RANGE [0] ::1 1 1 1
output: <OK>

# get
input:  0/1 P4G_IPV6_SERVER_RANGE [0] ?
output: 0/1 P4G_IPV6_SERVER_RANGE [0] ::1 1 1 1
```

P4G_IPV6_TRAFFIC_CLASS

```
# set
<module-index>/<port-index> P4G_IPV6_TRAFFIC_CLASS [<group_index>]
↪<traffic_class>

# get
<module-index>/<port-index> P4G_IPV6_TRAFFIC_CLASS [<group_index>] ?
```

Description

Configure the value of the traffic class field of the IPv6 header.

Actions

set, get

Parameters

`traffic_class`: *hex*, value of the traffic class field

Example

```
# set
input:  0/1 P4G_IPV6_TRAFFIC_CLASS [0] 0x00
output: <OK>

# get
input:  0/1 P4G_IPV6_TRAFFIC_CLASS [0] ?
output: 0/1 P4G_IPV6_TRAFFIC_CLASS [0] 0x00
```

P4G_L2_CLIENT_MAC

```
# set
<module-index>/<port-index> P4G_L2_CLIENT_MAC [<group_index>] <mac_
↪address> <mode>

# get
<module-index>/<port-index> P4G_L2_CLIENT_MAC [<group_index>] ?
```

Description

Configure the client MAC address. This is either a single static MAC address or an embedding of the four byte IPv4 address into the lower 4 bytes of the 6 byte MAC address.

Actions

set, get

Parameters

`mac_address`: *hex list*, the MAC address specified as hexadecimal

`mode`: *byte*, whether to embed the IP address in MAC

- `DONT_EMBED_IP = 0`
- `EMBED_IP = 1`

Example

```
# set
input:  0/1 P4G_L2_CLIENT_MAC [0] 0x000000000000 DONT_EMBED_IP
output: <OK>

# get
input:  0/1 P4G_L2_CLIENT_MAC [0] ?
output: 0/1 P4G_L2_CLIENT_MAC [0] 0x000000000000 DONT_EMBED_IP
```

P4G_L2_GW

```
# set
<module-index>/<port-index> P4G_L2_GW [<group_index>] <ipv4_address>
 ↪<mac_address>

# get
<module-index>/<port-index> P4G_L2_GW [<group_index>] ?
```

Description

Specify a default gateway for IPv4.

Actions

set, get

Parameters

`ipv4_address`: *address*, IPv5 address of the gateway

`mac_address`: *hex list*, the MAC address of the gateway

Example

```
# set
input:  0/1 P4G_L2_GW [0] 192.168.1.100 0x000000000000
output: <OK>

# get
input:  0/1 P4G_L2_GW [0] ?
output: 0/1 P4G_L2_GW [0] 192.168.1.100 0x000000000000
```

P4G_L2_IPV6_GW

```
# set
<module-index>/<port-index> P4G_L2_IPV6_GW [<group_index>] <ipv6_
 ↪address> <mac_address>

# get
<module-index>/<port-index> P4G_L2_IPV6_GW [<group_index>] ?
```

Description

Specify a default gateway for IPv6.

Actions

set, get

Parameters

`ipv6_address`: *string*, the 16 bytes of IPv6 address of gateway

`mac_address`: *hex list*, the MAC address of the gateway

Example

```
# set
input:  0/1 P4G_L2_IPV6_GW [0] ::1 0x000000000000
output: <OK>

# get
input:  0/1 P4G_L2_IPV6_GW [0] ?
output: 0/1 P4G_L2_IPV6_GW [0] ::1 0x000000000000
```

P4G_L2_SERVER_MAC

```
# set
<module-index>/<port-index> P4G_L2_SERVER_MAC [<group_index>] <mac_
↪address> <mode>

# get
<module-index>/<port-index> P4G_L2_SERVER_MAC [<group_index>] ?
```

Description

Configure the server MAC address. This is either a single static MAC address or an embedding of the four byte IPv4 address into the lower 4 bytes of the 6 byte MAC address.

Actions

set, get

Parameters

`mac_address`: *hex list*, the MAC address specified as hexadecimal

`mode`: *byte*, whether to embed the IP address in MAC

- `DONT_EMBED_IP = 0`
- `EMBED_IP = 1`

Example

```
# set
input:  0/1 P4G_L2_SERVER_MAC [0] 0x000000000000 DONT_EMBED_IP
output: <OK>

# get
input:  0/1 P4G_L2_SERVER_MAC [0] ?
output: 0/1 P4G_L2_SERVER_MAC [0] 0x000000000000 DONT_EMBED_IP
```

P4G_L2_USE_ADDRESS_RES

```
# set
<module-index>/<port-index> P4G_L2_USE_ADDRESS_RES [<group_index>] <is_
↪enabled>

# get
<module-index>/<port-index> P4G_L2_USE_ADDRESS_RES [<group_index>] ?
```

Description

Specify whether to use ARP and NDP to resolve hardware (MAC) addresses in the `pre_run` phase.

Actions

set, get

Parameters

is_enabled: *byte*, **specifying whether to use ARP and NDP to resolve hardware (MAC) addresses.**

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_L2_USE_ADDRESS_RES [0] NO
output: <OK>

# get
input:  0/1 P4G_L2_USE_ADDRESS_RES [0] ?
output: 0/1 P4G_L2_USE_ADDRESS_RES [0] NO
```

P4G_L2_USE_GW

```
# set
<module-index>/<port-index> P4G_L2_USE_GW [<group_index>] <is_enabled>

# get
<module-index>/<port-index> P4G_L2_USE_GW [<group_index>] ?
```

Description

Specify whether to use the resolved default gateway's MAC address as the destination MAC address in the packets.

Actions

set, get

Parameters

is_enabled: *byte*, **specifying whether to use gateway's MAC address as the destination MAC address in the packets.**

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_L2_USE_GW [0] NO
output: <OK>

# get
input:  0/1 P4G_L2_USE_GW [0] ?
output: 0/1 P4G_L2_USE_GW [0] NO
```

P4G_L4_PROTOCOL

```
# set
<module-index>/<port-index> P4G_L4_PROTOCOL [<group_index>] <protocol_
↪type>

# get
<module-index>/<port-index> P4G_L4_PROTOCOL [<group_index>] ?
```

Description

Specifies either TCP or UDP as Layer 4 protocol.

Actions

set, get

Parameters

protocol_type: *byte*, **the Layer 4 protocol**

- TCP = 0
- UDP = 1

Example

```
# set
input:  0/1 P4G_L4_PROTOCOL [0] TCP
output: <OK>

# get
input:  0/1 P4G_L4_PROTOCOL [0] ?
output: 0/1 P4G_L4_PROTOCOL [0] TCP
```

P4G_LP_SHAPE

```
# set
<module-index>/<port-index> P4G_LP_SHAPE [<group_index>] <star_time>
 ↪<rampup_duration> <steady_duration> <rampdown_duration>

# get
<module-index>/<port-index> P4G_LP_SHAPE [<group_index>] ?
```

Description

Specifies a load profile time duration. Time is measured from the beginning of the test when P4G_TRAFFIC is set to ON.

Actions

set, get

Parameters

`star_time`: *integer*, ramp-up start time

`rampup_duration`: *integer*, ramp-up phase duration

`steady_duration`: *integer*, steady phase duration

`rampdown_duration`: *integer*, ramp-down phase duration

Example

```
# set
input:  0/1 P4G_LP_SHAPE [0] 1 1 1 1
output: <OK>

# get
input:  0/1 P4G_LP_SHAPE [0] ?
output: 0/1 P4G_LP_SHAPE [0] 1 1 1 1
```

P4G_LP_TIME_SCALE

```
# set
<module-index>/<port-index> P4G_LP_TIME_SCALE [<group_index>]
↪<timescale>

# get
<module-index>/<port-index> P4G_LP_TIME_SCALE [<group_index>] ?
```

Description

Specifies the time scale of the load profile.

Actions

set, get

Parameters

timescale: *byte*, **specifying the time scale.**

- MSECS = 0
- SECONDS = 1
- MINUTES = 2
- HOURS = 3

Example

```
# set
input:  0/1 P4G_LP_TIME_SCALE [0] MSECS
output: <OK>

# get
input:  0/1 P4G_LP_TIME_SCALE [0] ?
output: 0/1 P4G_LP_TIME_SCALE [0] MSECS
```

P4G_NAT

```
# set
<module-index>/<port-index> P4G_NAT [<group_index>] <on_off>

# get
<module-index>/<port-index> P4G_NAT [<group_index>] ?
```

Description

Specify whether to support DUT Source NAT functionality. NAT should be enabled on both Client and Server ports that belong to the same CG.

Actions

set, get

Parameters

on_off: *byte*, **specifying whether to enable Source NAT support**

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P4G_NAT [0] OFF
output: <OK>

# get
input:  0/1 P4G_NAT [0] ?
output: 0/1 P4G_NAT [0] OFF
```

P4G_PAYLOAD_HIST_CONF

```
# set
<module-index>/<port-index> P4G_PAYLOAD_HIST_CONF [<group_index>]
 ↪<start> <interval>

# get
<module-index>/<port-index> P4G_PAYLOAD_HIST_CONF [<group_index>] ?
```

Description

Sets the start value and the interval size for the payload histograms.

Actions

set, get

Parameters

`start`: *long integer*, start value of first histogram interval in bytes

`interval`: *long integer*, histogram interval size in bytes

Example

```
# set
input:  0/1 P4G_PAYLOAD_HIST_CONF [0] 1 1
output: <OK>

# get
input:  0/1 P4G_PAYLOAD_HIST_CONF [0] ?
output: 0/1 P4G_PAYLOAD_HIST_CONF [0] 1 1
```

P4G_RAW_BURSTY_CONF

```
# set
<module-index>/<port-index> P4G_RAW_BURSTY_CONF [<group_index>]
 ↪<active_duration> <inactive_duration>

# get
<module-index>/<port-index> P4G_RAW_BURSTY_CONF [<group_index>] ?
```

Description

Specifies active and inactive periods of bursty transmission in milliseconds. The burst period starts with the active part.

Actions

set, get

Parameters

`active_duration`: *integer*, specifies the duration in milliseconds of the active part of the burst period.

`inactive_duration`: *integer*, specifies the duration in milliseconds of the inactive part of the burst period.

Example

```
# set
input:  0/1 P4G_RAW_BURSTY_CONF [0] 1 1
output: <OK>

# get
input:  0/1 P4G_RAW_BURSTY_CONF [0] ?
output: 0/1 P4G_RAW_BURSTY_CONF [0] 1 1
```

P4G_RAW_BURSTY_TX

```
# set
<module-index>/<port-index> P4G_RAW_BURSTY_TX [<group_index>] <bursty>

# get
<module-index>/<port-index> P4G_RAW_BURSTY_TX [<group_index>] ?
```

Description

Enables or disables bursty transmission.

Actions

set, get

Parameters

bursty: *byte*, **whether bursty transmission is on or off.**

- OFF = 0
- ON = 1

Example

```
# set
input:  0/1 P4G_RAW_BURSTY_TX [0] OFF
output: <OK>

# get
input:  0/1 P4G_RAW_BURSTY_TX [0] ?
output: 0/1 P4G_RAW_BURSTY_TX [0] OFF
```

P4G_RAW_CLOSE_CONN

```
# set
<module-index>/<port-index> P4G_RAW_CLOSE_CONN [<group_index>] <who_
↪close>

# get
<module-index>/<port-index> P4G_RAW_CLOSE_CONN [<group_index>] ?
```

Description

Specify how to close TCP connection when all payload has been transmitted.

In raw test scenario DOWNLOAD, the server can close the connection, when all payload has been transmitted.

In raw test scenario UPLOAD, the client can close the connection, when all payload has been transmitted. In any case, both server and client CGs must be configured with the same value of this parameter.

In raw test scenario BOTH (bidirectional), this parameter is N/A and will be ignored.

In a transaction scenario, where *P4G_RAW_HAS_DOWNLOAD_REQ* is set to YES, both client and server can close the connection, when the last transaction has been completed.

When *P4G_RAW_CONN_INCARNATION* is set to `IMMORTAL` or `REINCARNATE`, and this command is set to `NONE`, connections will be closed after 'connection lifetime', set by *P4G_RAW_CONN_LIFETIME*.

Note: This parameter is N/A when *P4G_L4_PROTOCOL* is configured as UDP.

Actions

set, get

Parameters

`who_close`: `WhoClose`, specifying how to close TCP connection

- `NONE = 0`
- `CLIENT = 1`
- `SERVER = 2`

Example

```
# set
input:  0/1 P4G_RAW_CLOSE_CONN [0] NONE
output: <OK>

# get
input:  0/1 P4G_RAW_CLOSE_CONN [0] ?
output: 0/1 P4G_RAW_CLOSE_CONN [0] NONE
```

P4G_RAW_CONN_INCARNATION

```
# set
<module-index>/<port-index> P4G_RAW_CONN_INCARNATION [<group_index>]
↪<mode>

# get
<module-index>/<port-index> P4G_RAW_CONN_INCARNATION [<group_index>] ?
```

Description

Defines the lifecycle of a connection and how new connections should be established as old connections are closed.

Actions

set, get

Parameters

mode: *byte*, **connection lifecycle mode**

- ONCE = 0
- IMMORTAL = 1
- REINCARNATE = 2

Example

```
# set
input:  0/1 P4G_RAW_CONN_INCARNATION [0] ONCE
output: <OK>

# get
input:  0/1 P4G_RAW_CONN_INCARNATION [0] ?
output: 0/1 P4G_RAW_CONN_INCARNATION [0] ONCE
```

P4G_RAW_CONN_LIFETIME

```
# set
<module-index>/<port-index> P4G_RAW_CONN_LIFETIME [<group_index>]
↪<timescale> <lifetime>

# get
<module-index>/<port-index> P4G_RAW_CONN_LIFETIME [<group_index>] ?
```

Description

Defines the lifetime of a connection, when *P4G_RAW_CONN_INCARNATION`* is set to `IMMORTAL` or `REINCARNATE`.

Actions

set, get

Parameters

`timescale`: *byte*, specifying the time scale

- `MSECS = 0`
- `SECONDS = 1`
- `MINUTES = 2`
- `HOURS = 3`

`lifetime`: *integer*, time from a connection is established until it will be closed.

Example

```
# set
input:  0/1 P4G_RAW_CONN_LIFETIME [0] MSECS 1
output: <OK>

# get
input:  0/1 P4G_RAW_CONN_LIFETIME [0] ?
output: 0/1 P4G_RAW_CONN_LIFETIME [0] MSECS 1
```

P4G_RAW_CONN_REPETITIONS

```
# set
<module-index>/<port-index> P4G_RAW_CONN_REPETITIONS [<group_index>]
↪<mode> <repetition_count>

# get
<module-index>/<port-index> P4G_RAW_CONN_REPETITIONS [<group_index>] ?
```

Description

Defines how many times a new connection should be created, after an old connection has been closed, when P4G_RAW_CONN_INCARNATION is set to IMMORTAL or REINCARNATE.

Actions

set, get

Parameters

mode: *byte*, **repetition mode.**

- INFINITE = 0
- FINITE = 1

repetition_count: *integer*, number of repetitions

Example

```
# set
input:  0/1 P4G_RAW_CONN_REPETITIONS [0] INFINITE 1
output: <OK>

# get
input:  0/1 P4G_RAW_CONN_REPETITIONS [0] ?
output: 0/1 P4G_RAW_CONN_REPETITIONS [0] INFINITE 1
```

P4G_RAW_DOWNLOAD_REQUEST

```
# set
<module-index>/<port-index> P4G_RAW_DOWNLOAD_REQUEST [<group_index>]
↪<length> <content>

# get
<module-index>/<port-index> P4G_RAW_DOWNLOAD_REQUEST [<group_index>] ?
```

Description

Specify the content of the download request sent by the client and expected by the server as hex bytes.

Note: This parameter is N/A when P4G_L4_PROTOCOL is configured as UDP.

Actions

set, get

Parameters

`length`: *integer*, specifying the number of bytes to write. Maximum request length is 1024 bytes.

`content`: *hex list*, specifying the request content.

Example

```
# set
input:  0/1 P4G_RAW_DOWNLOAD_REQUEST [0] 1␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B0001020
output: <OK>

# get
input:  0/1 P4G_RAW_DOWNLOAD_REQUEST [0] ?
output: 0/1 P4G_RAW_DOWNLOAD_REQUEST [0] 1␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B0001020
```

P4G_RAW_HAS_DOWNLOAD_REQ

```
# set
<module-index>/<port-index> P4G_RAW_HAS_DOWNLOAD_REQ [<group_index>]
↪<on_off>

# get
<module-index>/<port-index> P4G_RAW_HAS_DOWNLOAD_REQ [<group_index>] ?
```

Description

Specify whether the server waits for a request from the client before it starts transmitting.

Note: This parameter is N/A when P4G_L4_PROTOCOL is configured as UDP.

Actions

set, get

Parameters

on_off: *byte*, whether the server waits for a request from the client before it starts transmitting.

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_RAW_HAS_DOWNLOAD_REQ [0] NO
output: <OK>

# get
input:  0/1 P4G_RAW_HAS_DOWNLOAD_REQ [0] ?
output: 0/1 P4G_RAW_HAS_DOWNLOAD_REQ [0] NO
```

P4G_RAW_PAYLOAD

```
# set
<module-index>/<port-index> P4G_RAW_PAYLOAD [<group_index>] <offset>
↪<length> <content>

# get
<module-index>/<port-index> P4G_RAW_PAYLOAD [<group_index>] ?
```

Description

Specify raw payload as hex bytes. This command can be called several times to build a custom payload.

Actions

set, get

Parameters

`offset`: *integer*, the offset in the payload buffer where data is to be written

`length`: *integer*, number of bytes to write

`content`: *hex list*, specifying the payload

Example

```
# set
input:  0/1 P4G_RAW_PAYLOAD [0] 1 1␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B00010203
output: <OK>

# get
input:  0/1 P4G_RAW_PAYLOAD [0] ?
output: 0/1 P4G_RAW_PAYLOAD [0] 1 1␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B00010203
```

P4G_RAW_PAYLOAD_REPEAT_LEN

```
# set
<module-index>/<port-index> P4G_RAW_PAYLOAD_REPEAT_LEN [<group_index>]
↪<length>

# get
<module-index>/<port-index> P4G_RAW_PAYLOAD_REPEAT_LEN [<group_index>]␣
↪?
```

Description

Specify the length of the raw payload, which is defined by one or more P4G_RAW_PAYLOAD commands, to repeat.

P4G_RAW_PAYLOAD_REPEAT_LEN number of bytes will be repeated until P4G_RAW_PAYLOAD_TOTAL_LEN bytes are transmitted on the connection.

Actions

set, get

Parameters

length: *integer*, the length of the raw payload to repeat

Example

```
# set
input:  0/1 P4G_RAW_PAYLOAD_REPEAT_LEN [0] 1
output: <OK>

# get
input:  0/1 P4G_RAW_PAYLOAD_REPEAT_LEN [0] ?
output: 0/1 P4G_RAW_PAYLOAD_REPEAT_LEN [0] 1
```

P4G_RAW_PAYLOAD_TOTAL_LEN

```
# set
<module-index>/<port-index> P4G_RAW_PAYLOAD_TOTAL_LEN [<group_index>]
 ↪<mode> <length>

# get
<module-index>/<port-index> P4G_RAW_PAYLOAD_TOTAL_LEN [<group_index>] ?
```

Description

Configure the total amount of payload to transmit on one connection.

Actions

set, get

Parameters

mode: *byte*, **generation mode.**

- `INFINITE = 0`
- `FINITE = 1`

`length`: *long integer*, size of the payload

Example

```
# set
input:  0/1 P4G_RAW_PAYLOAD_TOTAL_LEN [0] INFINITE 1
output: <OK>

# get
input:  0/1 P4G_RAW_PAYLOAD_TOTAL_LEN [0] ?
output: 0/1 P4G_RAW_PAYLOAD_TOTAL_LEN [0] INFINITE 1
```

P4G_RAW_PAYLOAD_TYPE

```
# set
<module-index>/<port-index> P4G_RAW_PAYLOAD_TYPE [<group_index>] <gen_
↪method>

# get
<module-index>/<port-index> P4G_RAW_PAYLOAD_TYPE [<group_index>] ?
```

Description

Specify the payload generation method.

Actions

set, get

Parameters

gen_method: *byte*, **payload generation method**

- FIXED = 0
- INCREMENT = 1
- RANDOM = 2
- LONGRANDOM = 3

Example

```
# set
input:  0/1 P4G_RAW_PAYLOAD_TYPE [0] FIXED
output: <OK>

# get
input:  0/1 P4G_RAW_PAYLOAD_TYPE [0] ?
output: 0/1 P4G_RAW_PAYLOAD_TYPE [0] FIXED
```

P4G_RAW_REQUEST_REPEAT

```
# set
<module-index>/<port-index> P4G_RAW_REQUEST_REPEAT [<group_index>]
 ↪<mode> <repeat>

# get
<module-index>/<port-index> P4G_RAW_REQUEST_REPEAT [<group_index>] ?
```

Description

Specify the number of request/response transactions to perform - if P4G_RAW_HAS_DOWNLOAD_REQ is set to YES.

Note: This parameter is N/A when P4G_L4_PROTOCOL is configured as UDP.

Actions

set, get

Parameters

mode: *byte*, **specifying the transaction mode.**

- INFINITE = 0
- FINITE = 1

repeat: *integer*, number of request/response transactions to perform , if mode is FINITE.

Example

```
# set
input:  0/1 P4G_RAW_REQUEST_REPEAT [0] INFINITE 1
output: <OK>

# get
input:  0/1 P4G_RAW_REQUEST_REPEAT [0] ?
output: 0/1 P4G_RAW_REQUEST_REPEAT [0] INFINITE 1
```

P4G_RAW_RX_PAYLOAD_LEN

```
# set
<module-index>/<port-index> P4G_RAW_RX_PAYLOAD_LEN [<group_index>]
 ↪<mode> <length>

# get
<module-index>/<port-index> P4G_RAW_RX_PAYLOAD_LEN [<group_index>] ?
```

Description

Specify the length of the payload the Client should expect to receive before sending the next download request to the Server. Should be configured identical to the P4G_RAW_PAYLOAD_TOTAL_LEN for the Server. If mode is set to INFINITE, effectively no request/response repetitions will be performed.

Note: This parameter is N/A when P4G_L4_PROTOCOL is configured as UDP.

Actions

set, get

Parameters

mode: *byte*, **specifying the payload length mode**

- INFINITE = 0
- FINITE = 1

length: *long integer*, number of payload bytes the client should receive before sending the next request, if mode is FINITE.

Example

```
# set
input:  0/1 P4G_RAW_RX_PAYLOAD_LEN [0] INFINITE 1
output: <OK>

# get
input:  0/1 P4G_RAW_RX_PAYLOAD_LEN [0] ?
output: 0/1 P4G_RAW_RX_PAYLOAD_LEN [0] INFINITE 1
```

P4G_RAW_TEST_SCENARIO

```
# set
<module-index>/<port-index> P4G_RAW_TEST_SCENARIO [<group_index>]
 ↪<scenario>

# get
<module-index>/<port-index> P4G_RAW_TEST_SCENARIO [<group_index>] ?
```

Description

Configure the traffic direction scenario for RAW mode.

Actions

set, get

Parameters

scenario: *byte*, **traffic scenario**

- DOWNLOAD = 0
- UPLOAD = 1
- BOTH = 2
- ECHO = 3

Example

```
# set
input:  0/1 P4G_RAW_TEST_SCENARIO [0] DOWNLOAD
output: <OK>

# get
input:  0/1 P4G_RAW_TEST_SCENARIO [0] ?
output: 0/1 P4G_RAW_TEST_SCENARIO [0] DOWNLOAD
```

P4G_RAW_TX_DURING_RAMP

```
# set
<module-index>/<port-index> P4G_RAW_TX_DURING_RAMP [<group_index>]
 ↪<should_close_conn_ramp_up> <should_close_conn_ramp_down>

# get
<module-index>/<port-index> P4G_RAW_TX_DURING_RAMP [<group_index>] ?
```

Description

Specify if TCP payload transmission should take place during ramp-up and ramp-down.

Note: For UDP connections payload transmission will always take place during ramp-up and ramp-down, and this parameter is therefore N/A.

Actions

set, get

Parameters

should_close_conn_ramp_up: *byte*, whether TCP payload transmission should take place during ramp-up.

- NO = 0
- YES = 1

should_close_conn_ramp_down: *byte*, whether TCP payload transmission should take place during ramp-down.

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_RAW_TX_DURING_RAMP [0] NO NO
output: <OK>

# get
input:  0/1 P4G_RAW_TX_DURING_RAMP [0] ?
output: 0/1 P4G_RAW_TX_DURING_RAMP [0] NO NO
```

P4G_RAW_TX_TIME_OFFSET

```
# set
<module-index>/<port-index> P4G_RAW_TX_TIME_OFFSET [<group_index>]
 ↪<start_offset> <stop_offset>

# get
<module-index>/<port-index> P4G_RAW_TX_TIME_OFFSET [<group_index>] ?
```

Description

Specify a time offset to the transmit start and stop time, if `P4G_RAW_TX_DURING_RAMP` is set to `NO` for ramp-up and ramp-down respectively.

Actions

set, get

Parameters

`start_offset`: *integer*, specify time in milliseconds from ramp-up has completed to start of payload transmit.

`stop_offset`: *integer*, specify time in milliseconds from stop of payload transmit to start of ramp-down.

Example

```
# set
input:  0/1 P4G_RAW_TX_TIME_OFFSET [0] 1 1
output: <OK>

# get
input:  0/1 P4G_RAW_TX_TIME_OFFSET [0] ?
output: 0/1 P4G_RAW_TX_TIME_OFFSET [0] 1 1
```

P4G_RAW_UTILIZATION

```
# set
<module-index>/<port-index> P4G_RAW_UTILIZATION [<group_index>]
 ↪<utilization>

# get
<module-index>/<port-index> P4G_RAW_UTILIZATION [<group_index>] ?
```

Description

Specify the link layer bandwidth utilization for all the generated traffic from the specified Raw CG.

Actions

set, get

Parameters

`utilization`: *integer*, utilization specified in ppm.

Example

```
# set
input:  0/1 P4G_RAW_UTILIZATION [0] 1
output: <OK>

# get
input:  0/1 P4G_RAW_UTILIZATION [0] ?
output: 0/1 P4G_RAW_UTILIZATION [0] 1
```

P4G_RECALC_PAYLOAD_HIST

```
# set
<module-index>/<port-index> P4G_RECALC_PAYLOAD_HIST [<group_index>]
```

Description

Recalculates connection payload histograms (retrieved with: P4G_TCP_RX_TOTAL_BYTES_HIST, P4G_TCP_RX_GOOD_BYTES_HIST, P4G_TCP_TX_TOTAL_BYTES_HIST and P4G_TCP_TX_GOOD_BYTES_HIST). Used in case payload histogram configuration has been changed (using P4G_PAYLOAD_HIST_CONF)

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_RECALC_PAYLOAD_HIST [0]
output: <OK>
```

P4G_RECALC_TIME_HIST

```
# set
<module-index>/<port-index> P4G_RECALC_TIME_HIST [<group_index>]
```

Description

Recalculates connection time histograms (retrieved with: P4G_TCP_ESTABLISH_HIST and P4G_TCP_CLOSE_HIST). Used in case time histogram configuration has been changed (using P4G_TIME_HIST_CONF).

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_RECALC_TIME_HIST [0]
output: <OK>
```

P4G_RECALC_TRANSACTION_HIST

```
# set
<module-index>/<port-index> P4G_RECALC_TRANSACTION_HIST [<group_index>]
```

Description

Recalculates transaction histograms (retrieved with: P4G_APP_TRANSACTION_HIST). Used in case transaction histogram configuration has been changed (using P4G_TRANSACTION_HIST_CONF)

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_RECALC_TRANSACTION_HIST [0]
output: <OK>
```

P4G_REPLAY_FILE_CLEAR

```
# set
<module-index>/<port-index> P4G_REPLAY_FILE_CLEAR [<replay_file_index>]
```

Description

Clears a Replay File index, so no Replay File is configured for that index.

Actions

set

Parameters

Example

```
# set
input:  0/1 P4G_REPLAY_FILE_CLEAR [0]
output: <OK>
```

P4G_REPLAY_FILE_INDICES

```
# get
<module-index>/<port-index> P4G_REPLAY_FILE_INDICES [<group_index>] ?
```

Description

Returns an index list of configured Replay Files for this CG. These are the Replay File Index that are used for `P4G_REPLAY_FILE_NAME` and `P4G_REPLAY_FILE_CLEAR` commands. More than one Replay File can be configured for a CG. When configuring a Replay File for a CG, it must have an index.

Actions

get

Parameters

`replay_file_indices`: *integer list*, an index list of configured Replay Files for this CG.

Example

```
# get
input:  0/1 P4G_REPLAY_FILE_INDICES [0] ?
output: 0/1 P4G_REPLAY_FILE_INDICES [0] 0 1
```

P4G_REPLAY_FILE_NAME

```
# set
<module-index>/<port-index> P4G_REPLAY_FILE_NAME [<group_index>,
↪<replay_file_index>] <file_name>

# get
<module-index>/<port-index> P4G_REPLAY_FILE_NAME [<group_index>,
↪<replay_file_index>] ?
```

Description

More than one Replay File can be configured for a CG. When configuring a Replay File for a CG, it must have an index. The indices at which Replay Files are configured does not have to be continuous.

Actions

set, get

Parameters

`file_name`: *string*, file name (including relative path and excluding the `.bson` extension).

Example

```
# set
input:  0/1 P4G_REPLAY_FILE_NAME [0, 0]␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B0001020
output: <OK>

# get
input:  0/1 P4G_REPLAY_FILE_NAME [0, 0] ?
output: 0/1 P4G_REPLAY_FILE_NAME [0, 0]␣
↪0x000102030405060708090A0B000102030405060708090A0B000102030405060708090A0B0001020
```

P4G_REPLAY_USER_INCARNATION

```
# set
<module-index>/<port-index> P4G_REPLAY_USER_INCARNATION [<group_index>
↪] <mode>

# get
<module-index>/<port-index> P4G_REPLAY_USER_INCARNATION [<group_index>
↪] ?
```

Description

Defines the lifecycle mode of a user and its connections, and how new users should be established as old connections are closed.

Actions

set, get

Parameters

mode: *byte*, **defines the lifecycle mode of connections.**

- ONCE = 0
- IMMORTAL = 1
- REINCARNATE = 2

Example

```
# set
input:  0/1 P4G_REPLAY_USER_INCARNATION [0] ONCE
output: <OK>

# get
input:  0/1 P4G_REPLAY_USER_INCARNATION [0] ?
output: 0/1 P4G_REPLAY_USER_INCARNATION [0] ONCE
```

P4G_REPLAY_USER_REPETITIONS

```
# set
<module-index>/<port-index> P4G_REPLAY_USER_REPETITIONS [<group_index>
↪] <mode> <repetition_count>

# get
<module-index>/<port-index> P4G_REPLAY_USER_REPETITIONS [<group_index>
↪] ?
```

Description

Defines how many times a new user should be created after an old user has been destroyed, when `P4G_REPLAY_USER_INCARNATION` is set to `IMMORTAL` or `REINCARNATE`.

Actions

set, get

Parameters

`mode`: *byte*, the repetition mode

- `INFINITE = 0`
- `FINITE = 1`

`repetition_count`: *integer*, number of repetitions

Example

```
# set
input:  0/1 P4G_REPLAY_USER_REPETITIONS [0] INFINITE 1
output: <OK>

# get
input:  0/1 P4G_REPLAY_USER_REPETITIONS [0] ?
output: 0/1 P4G_REPLAY_USER_REPETITIONS [0] INFINITE 1
```

P4G_REPLAY_UTILIZATION

```
# set
<module-index>/<port-index> P4G_REPLAY_UTILIZATION [<group_index>]
↪<utilization>

# get
<module-index>/<port-index> P4G_REPLAY_UTILIZATION [<group_index>] ?
```

Description

Specify the link layer bandwidth utilization for all the generated traffic from the specified Replay CG.

Actions

set, get

Parameters

`utilization`: *integer*, utilization specified in ppm.

Example

```
# set
input:  0/1 P4G_REPLAY_UTILIZATION [0] 1
output: <OK>

# get
input:  0/1 P4G_REPLAY_UTILIZATION [0] ?
output: 0/1 P4G_REPLAY_UTILIZATION [0] 1
```

P4G_ROLE

```
# set
<module-index>/<port-index> P4G_ROLE [<group_index>] <role>

# get
<module-index>/<port-index> P4G_ROLE [<group_index>] ?
```

Description

Specifies the client or server role for this CG. A server passively waits for the clients to establish connections.

Actions

set, get

Parameters

role: **Role, the role of the CG.**

- CLIENT = 0
- SERVER = 1

Example

```
# set
input:  0/1 P4G_ROLE [0] CLIENT
output: <OK>

# get
input:  0/1 P4G_ROLE [0] ?
output: 0/1 P4G_ROLE [0] CLIENT
```

P4G_SERVER_RANGE

```
# set
<module-index>/<port-index> P4G_SERVER_RANGE [<group_index>] <ipv4_
↪address> <address_count> <start_port> <port_count>

# get
<module-index>/<port-index> P4G_SERVER_RANGE [<group_index>] ?
```

Description

Specifies a number of server sockets (ip address, port number)

Actions

set, get

Parameters

`ipv4_address`: *address*, the start IP address of the address range

`address_count`: *integer*, the number of IP addresses

`start_port`: *integer*, the starting port number of the port range

`port_count`: *integer*, the number of ports

Example

```
# set
input:  0/1 P4G_SERVER_RANGE [0] 192.168.1.100 1 1 1
output: <OK>

# get
input:  0/1 P4G_SERVER_RANGE [0] ?
output: 0/1 P4G_SERVER_RANGE [0] 192.168.1.100 1 1 1
```

P4G_TCP_ACK_FREQUENCY

```
# set
<module-index>/<port-index> P4G_TCP_ACK_FREQUENCY [<group_index>]
 ↪<packets_before_ack>

# get
<module-index>/<port-index> P4G_TCP_ACK_FREQUENCY [<group_index>] ?
```

Description

Number of received packets before a pure-ACK is sent.

Actions

set, get

Parameters

`packets_before_ack`: *integer*, number of received packets before an ACK is sent, range between 1 and 255, default 1. When set to 1, every packet is ACKed.

Example

```
# set
input:  0/1 P4G_TCP_ACK_FREQUENCY [0] 1
output: <OK>

# get
input:  0/1 P4G_TCP_ACK_FREQUENCY [0] ?
output: 0/1 P4G_TCP_ACK_FREQUENCY [0] 1
```

P4G_TCP_ACK_TIMEOUT

```
# set
<module-index>/<port-index> P4G_TCP_ACK_TIMEOUT [<group_index>] <ack_
↪timeout>

# get
<module-index>/<port-index> P4G_TCP_ACK_TIMEOUT [<group_index>] ?
```

Description

Delayed ACK timeout in microsecondsA pure ACK for the last RX packet will be sent after `P4G_TCP_ACK_TIMEOUT` microseconds in case it cannot be sent by other means, ie. a number of packets received since last ACK is less than `P4G_TCP_ACK_FREQUENCY` and there is no TX packets to sent (to piggy-back an ACK)

Actions

set, get

Parameters

`ack_timeout`: *integer*, timeout value in microseconds, default 200000.

Example

```
# set
input:  0/1 P4G_TCP_ACK_TIMEOUT [0] 1
output: <OK>

# get
input:  0/1 P4G_TCP_ACK_TIMEOUT [0] ?
output: 0/1 P4G_TCP_ACK_TIMEOUT [0] 1
```

P4G_TCP_CLOSE_HIST

```
# get
<module-index>/<port-index> P4G_TCP_CLOSE_HIST [<group_index>] ?
```

Description

Returns a histogram over TCP connection close times, with start and interval values as configured by `P4G_TIME_HIST_CONF`.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over TCP connection close times

`min_connection_close_time`: *long integer*, a histogram over TCP connection close times

`max_connection_close_time`: *long integer*, a histogram over TCP connection close times

`avg_connection_close_time`: *long integer*, a histogram over TCP connection close times

`start`: *long integer*, a histogram over TCP connection close times

`interval`: *long integer*, a histogram over TCP connection close times

`bin_00`: *integer*, a histogram over TCP connection close times

`bin_01`: *integer*, a histogram over TCP connection close times

`bin_02`: *integer*, a histogram over TCP connection close times

`bin_03`: *integer*, a histogram over TCP connection close times

`bin_04`: *integer*, a histogram over TCP connection close times

`bin_05`: *integer*, a histogram over TCP connection close times

`bin_06`: *integer*, a histogram over TCP connection close times

`bin_07`: *integer*, a histogram over TCP connection close times

`bin_08`: *integer*, a histogram over TCP connection close times

`bin_09`: *integer*, a histogram over TCP connection close times

`bin_10`: *integer*, a histogram over TCP connection close times

`bin_11`: *integer*, a histogram over TCP connection close times

`bin_12`: *integer*, a histogram over TCP connection close times

`bin_13`: *integer*, a histogram over TCP connection close times

`bin_14`: *integer*, a histogram over TCP connection close times

`bin_15`: *integer*, a histogram over TCP connection close times

`bin_16`: *integer*, a histogram over TCP connection close times

`bin_17`: *integer*, a histogram over TCP connection close times

`bin_18`: *integer*, a histogram over TCP connection close times

`bin_19`: *integer*, a histogram over TCP connection close times

`bin_20`: *integer*, a histogram over TCP connection close times

`bin_21`: *integer*, a histogram over TCP connection close times

`bin_22`: *integer*, a histogram over TCP connection close times

`bin_23`: *integer*, a histogram over TCP connection close times

`bin_24`: *integer*, a histogram over TCP connection close times

`bin_25`: *integer*, a histogram over TCP connection close times

`bin_26`: *integer*, a histogram over TCP connection close times

`bin_27`: *integer*, a histogram over TCP connection close times

`bin_28`: *integer*, a histogram over TCP connection close times

`bin_29`: *integer*, a histogram over TCP connection close times

`bin_30`: *integer*, a histogram over TCP connection close times

bin_31: *integer*, a histogram over TCP connection close times

Example

```
# get
input:  0/1 P4G_TCP_CLOSE_HIST [0] ?
output: 0/1 P4G_TCP_CLOSE_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_CONGESTION_MODE

```
# set
<module-index>/<port-index> P4G_TCP_CONGESTION_MODE [<group_index>]
↪<congestion_type>

# get
<module-index>/<port-index> P4G_TCP_CONGESTION_MODE [<group_index>] ?
```

Description

Configure the TCP congestion control algorithm.

Actions

set, get

Parameters

congestion_type: *byte*, specifying congestion algorithm type

- NONE = 0
- RENO = 1
- NEW_RENO = 2

Example

```
# set
input:  0/1 P4G_TCP_CONGESTION_MODE [0] NONE
output: <OK>

# get
input:  0/1 P4G_TCP_CONGESTION_MODE [0] ?
output: 0/1 P4G_TCP_CONGESTION_MODE [0] NONE
```

P4G_TCP_DUP_THRES

```
# set
<module-index>/<port-index> P4G_TCP_DUP_THRES [<group_index>]
↪<threshold>

# get
<module-index>/<port-index> P4G_TCP_DUP_THRES [<group_index>] ?
```

Description

Configure the value of the TCP duplicate ACK threshold.

Actions

set, get

Parameters

`threshold`: *short integer*, duplicate ACK threshold - must be larger than 0

Example

```
# set
input:  0/1 P4G_TCP_DUP_THRES [0] 1
output: <OK>

# get
input:  0/1 P4G_TCP_DUP_THRES [0] ?
output: 0/1 P4G_TCP_DUP_THRES [0] 1
```

P4G_TCP_ERROR_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_ERROR_COUNTERS [<group_index>] ?
```

Description

Returns a list of TCP error counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of TCP error counters.

`ref_time`: *long integer*, a list of TCP error counters.

`rx_reset_count`: *long integer*, a list of TCP error counters.

`tx_reset_count`: *long integer*, a list of TCP error counters.

`window_full_count`: *long integer*, a list of TCP error counters.

`max_syn_retrans_count`: *long integer*, a list of TCP error counters.

`max_retrans_count`: *long integer*, a list of TCP error counters.

`local_reset_count`: *long integer*, a list of TCP error counters.

`peer_reset_count`: *long integer*, a list of TCP error counters.

`seg_not_send_count`: *long integer*, a list of TCP error counters.

`rx_zero_window_count`: *long integer*, a list of TCP error counters.

Example

```
# get
input:  0/1 P4G_TCP_ERROR_COUNTERS [0] ?
output: 0/1 P4G_TCP_ERROR_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4G_TCP_ESTABLISH_HIST

```
# get
<module-index>/<port-index> P4G_TCP_ESTABLISH_HIST [<group_index>] ?
```

Description

Returns a histogram over TCP connection establish times, with start and interval values as configured by `P4G_TIME_HIST_CONF`.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over TCP connection establish times

`min_connection_estab_time`: *long integer*, a histogram over TCP connection establish times

`max_connection_estab_time`: *long integer*, a histogram over TCP connection establish times

`avg_connection_estab_time`: *long integer*, a histogram over TCP connection establish times

`start`: *long integer*, a histogram over TCP connection establish times

`interval`: *long integer*, a histogram over TCP connection establish times

`bin_00`: *integer*, a histogram over TCP connection establish times

`bin_01`: *integer*, a histogram over TCP connection establish times

`bin_02`: *integer*, a histogram over TCP connection establish times

`bin_03`: *integer*, a histogram over TCP connection establish times

`bin_04`: *integer*, a histogram over TCP connection establish times

`bin_05`: *integer*, a histogram over TCP connection establish times

`bin_06`: *integer*, a histogram over TCP connection establish times

`bin_07`: *integer*, a histogram over TCP connection establish times

`bin_08`: *integer*, a histogram over TCP connection establish times

`bin_09`: *integer*, a histogram over TCP connection establish times

`bin_10`: *integer*, a histogram over TCP connection establish times

bin_11: *integer*, a histogram over TCP connection establish times

bin_12: *integer*, a histogram over TCP connection establish times

bin_13: *integer*, a histogram over TCP connection establish times

bin_14: *integer*, a histogram over TCP connection establish times

bin_15: *integer*, a histogram over TCP connection establish times

bin_16: *integer*, a histogram over TCP connection establish times

bin_17: *integer*, a histogram over TCP connection establish times

bin_18: *integer*, a histogram over TCP connection establish times

bin_19: *integer*, a histogram over TCP connection establish times

bin_20: *integer*, a histogram over TCP connection establish times

bin_21: *integer*, a histogram over TCP connection establish times

bin_22: *integer*, a histogram over TCP connection establish times

bin_23: *integer*, a histogram over TCP connection establish times

bin_24: *integer*, a histogram over TCP connection establish times

bin_25: *integer*, a histogram over TCP connection establish times

bin_26: *integer*, a histogram over TCP connection establish times

bin_27: *integer*, a histogram over TCP connection establish times

bin_28: *integer*, a histogram over TCP connection establish times

bin_29: *integer*, a histogram over TCP connection establish times

bin_30: *integer*, a histogram over TCP connection establish times

bin_31: *integer*, a histogram over TCP connection establish times

Example

```
# get
input:  0/1 P4G_TCP_ESTABLISH_HIST [0] ?
output: 0/1 P4G_TCP_ESTABLISH_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_ICWND_CALC_METHOD

```
# set
<module-index>/<port-index> P4G_TCP_ICWND_CALC_METHOD [<group_index>]
↪<method> <factor>

# get
<module-index>/<port-index> P4G_TCP_ICWND_CALC_METHOD [<group_index>] ?
```

Description

Select the algorithm to calculate the TCP initial congestion window (ICWND).

Actions

set, get

Parameters

method: *byte*, **specifying the algorithm**

- RFC5681 = 0
- RFC2581 = 1
- FIXED_FACTOR = 2

factor: *integer*, factor to multiply the senders MSS with, when method is set to FIXED_FACTOR. Otherwise the value is ignored.

Example

```
# set
input:  0/1 P4G_TCP_ICWND_CALC_METHOD [0] RFC5681 1
output: <OK>

# get
input:  0/1 P4G_TCP_ICWND_CALC_METHOD [0] ?
output: 0/1 P4G_TCP_ICWND_CALC_METHOD [0] RFC5681 1
```

P4G_TCP_ISSTHRESH

```
# set
<module-index>/<port-index> P4G_TCP_ISSTHRESH [<group_index>] <mode>
↪<threshold>

# get
<module-index>/<port-index> P4G_TCP_ISSTHRESH [<group_index>] ?
```

Description

Configure the TCP initial slow start threshold (ISSTHRESH).

Actions

set, get

Parameters

mode: *byte*, **specifying TCP initial slow start mode**

- AUTOMATIC = 0
- MANUAL = 1

threshold: *integer*, number of bytes, value ignored when mode is set to MANUAL

Example

```
# set
input:  0/1 P4G_TCP_ISSTHRESH [0] AUTOMATIC 1
output: <OK>

# get
input:  0/1 P4G_TCP_ISSTHRESH [0] ?
output: 0/1 P4G_TCP_ISSTHRESH [0] AUTOMATIC 1
```

P4G_TCP_MSS_MINMAX

```
# set
<module-index>/<port-index> P4G_TCP_MSS_MINMAX [<group_index>] <mss_
↪min> <mss_max>

# get
<module-index>/<port-index> P4G_TCP_MSS_MINMAX [<group_index>] ?
```

Description

Configure the min and max values of the range for MSS. Both values are included in the range. Relevant when P4G_TCP_MSS_TYPE is set to INCREMENT or RANDOM.

Actions

set, get

Parameters

`mss_min`: *integer*, minimum value of MSS

`mss_max`: *integer*, maximum value of MSS

Example

```
# set
input:  0/1 P4G_TCP_MSS_MINMAX [0] 1 1
output: <OK>

# get
input:  0/1 P4G_TCP_MSS_MINMAX [0] ?
output: 0/1 P4G_TCP_MSS_MINMAX [0] 1 1
```

P4G_TCP_MSS_TYPE

```
# set
<module-index>/<port-index> P4G_TCP_MSS_TYPE [<group_index>] <mss_type>

# get
<module-index>/<port-index> P4G_TCP_MSS_TYPE [<group_index>] ?
```

Description

Specifies the Maximum Segment size (MSS) type for a CG. The MSS can either be fixed size identical for all connections in the CG, incrementing or random. The individual MSS for a specific connection is always constant once the incrementing or random value has been created. Refer to `P4G_TCP_MSS_MINMAX` command for information on how to configure min and max values.

Actions

set, get

Parameters

`mss_type`: *byte*, specifying how MSS is set

- `FIXED = 0`
- `INCREMENT = 1`
- `RANDOM = 2`

Example

```
# set
input:  0/1 P4G_TCP_MSS_TYPE [0] FIXED
output: <OK>

# get
input:  0/1 P4G_TCP_MSS_TYPE [0] ?
output: 0/1 P4G_TCP_MSS_TYPE [0] FIXED
```

P4G_TCP_MSS_VALUE

```
# set
<module-index>/<port-index> P4G_TCP_MSS_VALUE [<group_index>] <mss>

# get
<module-index>/<port-index> P4G_TCP_MSS_VALUE [<group_index>] ?
```

Description

Configure the fixed MSS value. Relevant when P4G_TCP_MSS_TYPE is set to FIXED.

Actions

set, get

Parameters

`mss`: *integer*, the fixed value of MSS (in bytes)

Example

```
# set
input:  0/1 P4G_TCP_MSS_VALUE [0] 1
output: <OK>

# get
input:  0/1 P4G_TCP_MSS_VALUE [0] ?
output: 0/1 P4G_TCP_MSS_VALUE [0] 1
```

P4G_TCP_RETRANSMIT_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_RETRANSMIT_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of TCP retransmission counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of TCP retransmission counters.

`ref_time`: *long integer*, a list of TCP retransmission counters.

`rx_duplicate_ack_count`: *long integer*, a list of TCP retransmission counters.

`rx_ooo_segment_count`: *long integer*, a list of TCP retransmission counters.

`fast_retrans_event_count`: *long integer*, a list of TCP retransmission counters.

`fast_retrans_segment_count`: *long integer*, a list of TCP retransmission counters.

`rto_retrans_event_count`: *long integer*, a list of TCP retransmission counters.

`syn_retrans_count`: *long integer*, a list of TCP retransmission counters.

`fin_retrans_count`: *long integer*, a list of TCP retransmission counters.

Example

```
# get
input:  0/1 P4G_TCP_RETRANSMIT_COUNTERS [0] ?
output: 0/1 P4G_TCP_RETRANSMIT_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_TCP_RTO

```
# set
<module-index>/<port-index> P4G_TCP_RTO [<group_index>] <rto_type>
↪<retrans_timeout> <retry_count> <backoff>

# get
<module-index>/<port-index> P4G_TCP_RTO [<group_index>] ?
```

Description

Configure the value of the TCP retransmission timeout, max retries and max backoff.

Actions

set, get

Parameters

rto_type: *byte*, **specifying RTO type**

- STATIC = 0
- DYNAMIC = 1

retrans_timeout: *integer*, retransmission timeout [milliseconds] - must be larger than 0

retry_count: *short integer*, maximum retransmission retries - must be larger than 0

backoff: *short integer*, maximum retransmission backoff

Example

```
# set
input:  0/1 P4G_TCP_RTO [0] STATIC 1 1 1
output: <OK>

# get
input:  0/1 P4G_TCP_RTO [0] ?
output: 0/1 P4G_TCP_RTO [0] STATIC 1 1 1
```

P4G_TCP_RTO_MINMAX

```
# set
<module-index>/<port-index> P4G_TCP_RTO_MINMAX [<group_index>] <rto_
↪min> <rto_max>

# get
<module-index>/<port-index> P4G_TCP_RTO_MINMAX [<group_index>] ?
```

Description

Configure the min and max values of the TCP retransmission timeout, when rto type is set to dynamic. If the calculated rto fall outside the interval, the value is clamped to the min or max value.

Actions

set, get

Parameters

`rto_min`: *integer*, min retransmission timeout [us] - must be larger than 0 and less than max.

`rto_max`: *integer*, max retransmission timeout [us] - must be larger than 0 and greater than min.

Example

```
# set
input:  0/1 P4G_TCP_RTO_MINMAX [0] 1 1
output: <OK>

# get
input:  0/1 P4G_TCP_RTO_MINMAX [0] ?
output: 0/1 P4G_TCP_RTO_MINMAX [0] 1 1
```

P4G_TCP_RTO_PROLONGED_MODE

```
# set
<module-index>/<port-index> P4G_TCP_RTO_PROLONGED_MODE [<group_index>]
 ↪<mode> <timeout>

# get
<module-index>/<port-index> P4G_TCP_RTO_PROLONGED_MODE [<group_index>]␣
 ↪?
```

Description

Configure TCP retransmission prolonged mode. When enabled, TCP will, after exceeding max number of retransmission retries, continue trying retransmit until success, whereafter it will operate normally.

Actions

set, get

Parameters

mode: *byte*, specifying whether to enable/disable prolonged retransmission mode

- DISABLE = 0
- ENABLE = 1

timeout: *integer*, retransmission timeout in milliseconds, when prolonged mode is enabled. When mode is set to 0, the value of the timeout is ignored. When mode is set to 1, the value of the timeout may not be 0.

Example

```
# set
input:  0/1 P4G_TCP_RTO_PROLONGED_MODE [0] DISABLE 1
output: <OK>

# get
input:  0/1 P4G_TCP_RTO_PROLONGED_MODE [0] ?
output: 0/1 P4G_TCP_RTO_PROLONGED_MODE [0] DISABLE 1
```

P4G_TCP_RTT_VALUE

```
# get
<module-index>/<port-index> P4G_TCP_RTT_VALUE [<group_index>] ?
```

Description

Returns values that can be used to calculate the RTT value of all connections in a CG.

Actions

get

Parameters

`current_time`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

`ref_time`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

`local_rtt_sum`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

`local_rtt_count`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

`global_rtt_sum`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

`global_rtt_count`: *long integer*, values that can be used to calculate the RTT value of all connections in a CG.

Example

```
# get
input:  0/1 P4G_TCP_RTT_VALUE [0] ?
output: 0/1 P4G_TCP_RTT_VALUE [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4G_TCP_RX_GOOD_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TCP_RX_GOOD_BYTES_HIST [<group_index>]␣
↪?
```

Description

Returns a histogram over number of good TCP bytes received, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of good TCP bytes received

`min_byte_count`: *long integer*, a histogram over number of good TCP bytes received

`max_byte_count`: *long integer*, a histogram over number of good TCP bytes received

`avg`: *long integer*, a histogram over number of good TCP bytes received

`start`: *long integer*, a histogram over number of good TCP bytes received

`interval`: *long integer*, a histogram over number of good TCP bytes received

`bin_00`: *integer*, a histogram over number of good TCP bytes received

`bin_01`: *integer*, a histogram over number of good TCP bytes received

`bin_02`: *integer*, a histogram over number of good TCP bytes received

`bin_03`: *integer*, a histogram over number of good TCP bytes received

`bin_04`: *integer*, a histogram over number of good TCP bytes received

`bin_05`: *integer*, a histogram over number of good TCP bytes received

`bin_06`: *integer*, a histogram over number of good TCP bytes received

`bin_07`: *integer*, a histogram over number of good TCP bytes received

`bin_08`: *integer*, a histogram over number of good TCP bytes received

`bin_09`: *integer*, a histogram over number of good TCP bytes received

`bin_10`: *integer*, a histogram over number of good TCP bytes received

`bin_11`: *integer*, a histogram over number of good TCP bytes received

`bin_12`: *integer*, a histogram over number of good TCP bytes received

`bin_13`: *integer*, a histogram over number of good TCP bytes received

`bin_14`: *integer*, a histogram over number of good TCP bytes received

`bin_15`: *integer*, a histogram over number of good TCP bytes received

`bin_16`: *integer*, a histogram over number of good TCP bytes received

bin_17: *integer*, a histogram over number of good TCP bytes received

bin_18: *integer*, a histogram over number of good TCP bytes received

bin_19: *integer*, a histogram over number of good TCP bytes received

bin_20: *integer*, a histogram over number of good TCP bytes received

bin_21: *integer*, a histogram over number of good TCP bytes received

bin_22: *integer*, a histogram over number of good TCP bytes received

bin_23: *integer*, a histogram over number of good TCP bytes received

bin_24: *integer*, a histogram over number of good TCP bytes received

bin_25: *integer*, a histogram over number of good TCP bytes received

bin_26: *integer*, a histogram over number of good TCP bytes received

bin_27: *integer*, a histogram over number of good TCP bytes received

bin_28: *integer*, a histogram over number of good TCP bytes received

bin_29: *integer*, a histogram over number of good TCP bytes received

bin_30: *integer*, a histogram over number of good TCP bytes received

bin_31: *integer*, a histogram over number of good TCP bytes received

Example

```
# get
input:  0/1 P4G_TCP_RX_GOOD_BYTES_HIST [0] ?
output: 0/1 P4G_TCP_RX_GOOD_BYTES_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_RX_PACKET_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_RX_PACKET_COUNTERS [<group_index>]␣
↪?
```

Description

Returns a list of the TCP RX packet counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TCP RX packet counters.

`ref_time`: *long integer*, a list of the TCP RX packet counters.

`packet_count`: *long integer*, a list of the TCP RX packet counters.

`packet_per_second`: *long integer*, a list of the TCP RX packet counters.

Example

```
# get
input:  0/1 P4G_TCP_RX_PACKET_COUNTERS [0] ?
output: 0/1 P4G_TCP_RX_PACKET_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_TCP_RX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_RX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the TCP Rx payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TCP Rx payload counters.

`ref_time`: *long integer*, a list of the TCP Rx payload counters.

`total_byte_count`: *long integer*, a list of the TCP Rx payload counters.

`total_byte_per_second`: *long integer*, a list of the TCP Rx payload counters.

`good_byte_count`: *long integer*, a list of the TCP Rx payload counters.

`good_byte_per_second`: *long integer*, a list of the TCP Rx payload counters.

Example

```
# get
input:  0/1 P4G_TCP_RX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_TCP_RX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4G_TCP_RX_TOTAL_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TCP_RX_TOTAL_BYTES_HIST [<group_index>
↪] ?
```

Description

Returns a histogram over number of total TCP bytes received, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of total TCP bytes received

`min_byte_count`: *long integer*, a histogram over number of total TCP bytes received

`max_byte_count`: *long integer*, a histogram over number of total TCP bytes received

`avg_byte_count`: *long integer*, a histogram over number of total TCP bytes received

`start`: *long integer*, a histogram over number of total TCP bytes received

`interval`: *long integer*, a histogram over number of total TCP bytes received

`bin_00`: *integer*, a histogram over number of total TCP bytes received

`bin_01`: *integer*, a histogram over number of total TCP bytes received

`bin_02`: *integer*, a histogram over number of total TCP bytes received

`bin_03`: *integer*, a histogram over number of total TCP bytes received

`bin_04`: *integer*, a histogram over number of total TCP bytes received

`bin_05`: *integer*, a histogram over number of total TCP bytes received

`bin_06`: *integer*, a histogram over number of total TCP bytes received

`bin_07`: *integer*, a histogram over number of total TCP bytes received

`bin_08`: *integer*, a histogram over number of total TCP bytes received

`bin_09`: *integer*, a histogram over number of total TCP bytes received

`bin_10`: *integer*, a histogram over number of total TCP bytes received

`bin_11`: *integer*, a histogram over number of total TCP bytes received

`bin_12`: *integer*, a histogram over number of total TCP bytes received

`bin_13`: *integer*, a histogram over number of total TCP bytes received

`bin_14`: *integer*, a histogram over number of total TCP bytes received

`bin_15`: *integer*, a histogram over number of total TCP bytes received

`bin_16`: *integer*, a histogram over number of total TCP bytes received

`bin_17`: *integer*, a histogram over number of total TCP bytes received

`bin_18`: *integer*, a histogram over number of total TCP bytes received

`bin_19`: *integer*, a histogram over number of total TCP bytes received

`bin_20`: *integer*, a histogram over number of total TCP bytes received

bin_21: *integer*, a histogram over number of total TCP bytes received

bin_22: *integer*, a histogram over number of total TCP bytes received

bin_23: *integer*, a histogram over number of total TCP bytes received

bin_24: *integer*, a histogram over number of total TCP bytes received

bin_25: *integer*, a histogram over number of total TCP bytes received

bin_26: *integer*, a histogram over number of total TCP bytes received

bin_27: *integer*, a histogram over number of total TCP bytes received

bin_28: *integer*, a histogram over number of total TCP bytes received

bin_29: *integer*, a histogram over number of total TCP bytes received

bin_30: *integer*, a histogram over number of total TCP bytes received

bin_31: *integer*, a histogram over number of total TCP bytes received

Example

```
# get
input:  0/1 P4G_TCP_RX_TOTAL_BYTES_HIST [0] ?
output: 0/1 P4G_TCP_RX_TOTAL_BYTES_HIST [0] 1 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_STATE_CURRENT

```
# get
<module-index>/<port-index> P4G_TCP_STATE_CURRENT [<group_index>] ?
```

Description

Returns a list of the current TCP state counters. The counters returned corresponds the the following TCP states:

- CLOSED
- LISTEN
- SYN_SENT
- TCP_SYN_RCVD
- ESTABLISHED
- FIN_WAIT_1

- FIN_WAIT_2
- CLOSE_WAIT
- CLOSING
- LAST_ACK
- TIME_WAIT

Actions

get

Parameters

`current_time`: *long integer*, a list of the current TCP state counters

`ref_time`: *long integer*, a list of the current TCP state counters

`closed`: *long integer*, a list of the current TCP state counters

`listen`: *long integer*, a list of the current TCP state counters

`syn_sent`: *long integer*, a list of the current TCP state counters

`syn_rcvd`: *long integer*, a list of the current TCP state counters

`established`: *long integer*, a list of the current TCP state counters

`fin_wait_1`: *long integer*, a list of the current TCP state counters

`fin_wait_2`: *long integer*, a list of the current TCP state counters

`close_wait`: *long integer*, a list of the current TCP state counters

`closing`: *long integer*, a list of the current TCP state counters

`last_ack`: *long integer*, a list of the current TCP state counters

`time_wait`: *long integer*, a list of the current TCP state counters

Example

```
# get
input:  0/1 P4G_TCP_STATE_CURRENT [0] ?
output: 0/1 P4G_TCP_STATE_CURRENT [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_TCP_STATE_RATE

```
# get
<module-index>/<port-index> P4G_TCP_STATE_RATE [<group_index>] ?
```

Description

Returns a list of the TCP state rates measured in connections/second. The counters returned corresponds the the following TCP state rates:

- CLOSED
- LISTEN
- SYN_SENT
- TCP_SYN_RCVD
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSE_WAIT
- CLOSING
- LAST_ACK
- TIME_WAIT

Actions

get

Parameters

`current_time`: *long integer*, a list of the TCP state rates measured in connections/second

`ref_time`: *long integer*, a list of the TCP state rates measured in connections/second

`closed`: *long integer*, a list of the TCP state rates measured in connections/second

`listen`: *long integer*, a list of the TCP state rates measured in connections/second

`syn_sent`: *long integer*, a list of the TCP state rates measured in connections/second

`syn_rcvd`: *long integer*, a list of the TCP state rates measured in connections/second

`established`: *long integer*, a list of the TCP state rates measured in connections/second

`fin_wait_1`: *long integer*, a list of the TCP state rates measured in connections/second

`fin_wait_2`: *long integer*, a list of the TCP state rates measured in connections/second

`close_wait`: *long integer*, a list of the TCP state rates measured in connections/second

`closing`: *long integer*, a list of the TCP state rates measured in connections/second

`last_ack`: *long integer*, a list of the TCP state rates measured in connections/second

`time_wait`: *long integer*, a list of the TCP state rates measured in connections/second

Example

```
# get
input:  0/1 P4G_TCP_STATE_RATE [0] ?
output: 0/1 P4G_TCP_STATE_RATE [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_TCP_STATE_TOTAL

```
# get
<module-index>/<port-index> P4G_TCP_STATE_TOTAL [<group_index>] ?
```

Description

Returns a list of the total TCP state counters. The counters returned corresponds the the following TCP states:

- CLOSED
- LISTEN
- SYN_SENT
- TCP_SYN_RCVD
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSE_WAIT
- CLOSING
- LAST_ACK
- TIME_WAIT

Actions

get

Parameters

`current_time`: *long integer*, a list of the total TCP state counters

`ref_time`: *long integer*, a list of the total TCP state counters

`closed`: *long integer*, a list of the total TCP state counters

`listen`: *long integer*, a list of the total TCP state counters

`syn_sent`: *long integer*, a list of the total TCP state counters

`syn_rcvd`: *long integer*, a list of the total TCP state counters

`established`: *long integer*, a list of the total TCP state counters

`fin_wait_1`: *long integer*, a list of the total TCP state counters

`fin_wait_2`: *long integer*, a list of the total TCP state counters

`close_wait`: *long integer*, a list of the total TCP state counters

`closing`: *long integer*, a list of the total TCP state counters

`last_ack`: *long integer*, a list of the total TCP state counters

`time_wait`: *long integer*, a list of the total TCP state counters

Example

```
# get
input:  0/1 P4G_TCP_STATE_TOTAL [0] ?
output: 0/1 P4G_TCP_STATE_TOTAL [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_TCP_SYN_RTO

```
# set
<module-index>/<port-index> P4G_TCP_SYN_RTO [<group_index>] <retrans_
↪timeout> <retry_count> <backoff>

# get
<module-index>/<port-index> P4G_TCP_SYN_RTO [<group_index>] ?
```

Description

Configure the value of the TCP SYN retransmission timeout, max retries and max backoff.

Actions

set, get

Parameters

`retrans_timeout`: *integer*, SYN retransmission timeout [milliseconds] - must be larger than 0

`retry_count`: *short integer*, maximum SYN retransmission retries - must be larger than 0

`backoff`: *short integer*, maximum SYN retransmission backoff

Example

```
# set
input:  0/1 P4G_TCP_SYN_RTO [0] 1 1 1
output: <OK>

# get
input:  0/1 P4G_TCP_SYN_RTO [0] ?
output: 0/1 P4G_TCP_SYN_RTO [0] 1 1 1
```

P4G_TCP_TX_GOOD_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TCP_TX_GOOD_BYTES_HIST [<group_index>]␣
↪?
```

Description

Returns a histogram over number of good TCP bytes transmitted, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`conn`: *integer*, a histogram over number of good TCP bytes transmitted

`min`: *long integer*, a histogram over number of good TCP bytes transmitted

`max`: *long integer*, a histogram over number of good TCP bytes transmitted

`average`: *long integer*, a histogram over number of good TCP bytes transmitted

`start`: *long integer*, a histogram over number of good TCP bytes transmitted

`interval`: *long integer*, a histogram over number of good TCP bytes transmitted

`bin_00`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_01`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_02`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_03`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_04`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_05`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_06`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_07`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_08`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_09`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_10`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_11`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_12`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_13`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_14`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_15`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_16`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_17`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_18`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_19`: *integer*, a histogram over number of good TCP bytes transmitted

`bin_20`: *integer*, a histogram over number of good TCP bytes transmitted

bin_21: *integer*, a histogram over number of good TCP bytes transmitted

bin_22: *integer*, a histogram over number of good TCP bytes transmitted

bin_23: *integer*, a histogram over number of good TCP bytes transmitted

bin_24: *integer*, a histogram over number of good TCP bytes transmitted

bin_25: *integer*, a histogram over number of good TCP bytes transmitted

bin_26: *integer*, a histogram over number of good TCP bytes transmitted

bin_27: *integer*, a histogram over number of good TCP bytes transmitted

bin_28: *integer*, a histogram over number of good TCP bytes transmitted

bin_29: *integer*, a histogram over number of good TCP bytes transmitted

bin_30: *integer*, a histogram over number of good TCP bytes transmitted

bin_31: *integer*, a histogram over number of good TCP bytes transmitted

Example

```
# get
input:  0/1 P4G_TCP_TX_GOOD_BYTES_HIST [0] ?
output: 0/1 P4G_TCP_TX_GOOD_BYTES_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_TX_PACKET_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_TX_PACKET_COUNTERS [<group_index>]␣
↪?
```

Description

Returns a list of the TCP TX packet counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TCP TX packet counters.

`ref_time`: *long integer*, a list of the TCP TX packet counters.

`packet_count`: *long integer*, a list of the TCP TX packet counters.

`packet_per_second`: *long integer*, a list of the TCP TX packet counters.

Example

```
# get
input:  0/1 P4G_TCP_TX_PACKET_COUNTERS [0] ?
output: 0/1 P4G_TCP_TX_PACKET_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_TCP_TX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_TCP_TX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the TCP Tx payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TCP Tx payload counters.

`ref_time`: *long integer*, a list of the TCP Tx payload counters.

`total_byte_count`: *long integer*, a list of the TCP Tx payload counters.

`total_byte_per_second`: *long integer*, a list of the TCP Tx payload counters.

`good_byte_count`: *long integer*, a list of the TCP Tx payload counters.

`good_byte_per_second`: *long integer*, a list of the TCP Tx payload counters.

Example

```
# get
input:  0/1 P4G_TCP_TX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_TCP_TX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4G_TCP_TX_TOTAL_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TCP_TX_TOTAL_BYTES_HIST [<group_index>
↪] ?
```

Description

Returns a histogram over number of total TCP bytes transmitted, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of total TCP bytes transmitted

`min_byte_count`: *long integer*, a histogram over number of total TCP bytes transmitted

`max_byte_count`: *long integer*, a histogram over number of total TCP bytes transmitted

`avg_byte_count`: *long integer*, a histogram over number of total TCP bytes transmitted

`start`: *long integer*, a histogram over number of total TCP bytes transmitted

`interval`: *long integer*, a histogram over number of total TCP bytes transmitted

`bin_00`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_01`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_02`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_03`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_04`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_05`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_06`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_07`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_08`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_09`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_10`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_11`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_12`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_13`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_14`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_15`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_16`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_17`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_18`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_19`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_20`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_21`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_22`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_23`: *integer*, a histogram over number of total TCP bytes transmitted

`bin_24`: *integer*, a histogram over number of total TCP bytes transmitted

bin_25: *integer*, a histogram over number of total TCP bytes transmitted

bin_26: *integer*, a histogram over number of total TCP bytes transmitted

bin_27: *integer*, a histogram over number of total TCP bytes transmitted

bin_28: *integer*, a histogram over number of total TCP bytes transmitted

bin_29: *integer*, a histogram over number of total TCP bytes transmitted

bin_30: *integer*, a histogram over number of total TCP bytes transmitted

bin_31: *integer*, a histogram over number of total TCP bytes transmitted

Example

```
# get
input:  0/1 P4G_TCP_TX_TOTAL_BYTES_HIST [0] ?
output: 0/1 P4G_TCP_TX_TOTAL_BYTES_HIST [0] 1 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TCP_WINDOW_SCALING

```
# set
<module-index>/<port-index> P4G_TCP_WINDOW_SCALING [<group_index>] <on_
↪off> <factor>

# get
<module-index>/<port-index> P4G_TCP_WINDOW_SCALING [<group_index>] ?
```

Description

Enable window scaling for the CG. Note to use windows scaling it need to be enabled in both the client and server CG. .

Actions

set, get

Parameters

on_off: *byte*, specifying whether to enable window scaling or not

- NO = 0
- YES = 1

factor: *short integer*, default value is 0 and maximum value is 14 - ignored if window scaling is not enabled

Example

```
# set
input:  0/1 P4G_TCP_WINDOW_SCALING [0] NO 1
output: <OK>

# get
input:  0/1 P4G_TCP_WINDOW_SCALING [0] ?
output: 0/1 P4G_TCP_WINDOW_SCALING [0] NO 1
```

P4G_TCP_WINDOW_SIZE

```
# set
<module-index>/<port-index> P4G_TCP_WINDOW_SIZE [<group_index>]
↪<window_size>

# get
<module-index>/<port-index> P4G_TCP_WINDOW_SIZE [<group_index>] ?
```

Description

Configure the value of the TCP RWND.

Actions

set, get

Parameters

`window_size`: *integer*, RWND size in bytes

Example

```
# set
input:  0/1 P4G_TCP_WINDOW_SIZE [0] 1
output: <OK>

# get
input:  0/1 P4G_TCP_WINDOW_SIZE [0] ?
output: 0/1 P4G_TCP_WINDOW_SIZE [0] 1
```

P4G_TEST_APPLICATION

```
# set
<module-index>/<port-index> P4G_TEST_APPLICATION [<group_index>]
↪<behavior>

# get
<module-index>/<port-index> P4G_TEST_APPLICATION [<group_index>] ?
```

Description

Configure the application layer mode. This command affects whether TCP payload is generated.

- `NONE` means that TCP connections are created according to the client and server ranges, and ramped up/down as specified in the load profile. But no payload is transmitted.
- `RAW` differs from `NONE` in that it transmits payload when the TCP connection is established.
- `REPLAY` refers to PCAP replay.

Actions

set, get

Parameters

behavior: *byte*, **the application layer mode**

- NONE = 0
- RAW = 1
- REPLAY = 2

Example

```
# set
input:  0/1 P4G_TEST_APPLICATION [0] NONE
output: <OK>

# get
input:  0/1 P4G_TEST_APPLICATION [0] ?
output: 0/1 P4G_TEST_APPLICATION [0] NONE
```

P4G_TIME_HIST_CONF

```
# set
<module-index>/<port-index> P4G_TIME_HIST_CONF [<group_index>] <start>
↪<interval>

# get
<module-index>/<port-index> P4G_TIME_HIST_CONF [<group_index>] ?
```

Description

Sets the start value and the interval size for the time histograms (P4G_TCP_ESTABLISH_HIST and P4G_TCP_CLOSE_HIST).

Actions

set, get

Parameters

`start`: *long integer*, start value of first histogram interval in microseconds

`interval`: *long integer*, histogram interval size in microseconds

Example

```
# set
input:  0/1 P4G_TIME_HIST_CONF [0] 1 1
output: <OK>

# get
input:  0/1 P4G_TIME_HIST_CONF [0] ?
output: 0/1 P4G_TIME_HIST_CONF [0] 1 1
```

P4G_TLS_ALERT_FATAL_COUNTERS

```
# get
<module-index>/<port-index> P4G_TLS_ALERT_FATAL_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of TLS error counters. The counters returned corresponds the the following TLS warnings:

- close_notify
- unexpected_message
- bad_record_mac
- record_overflow
- decompression_failure
- handshake_failure
- bad_certificate
- unsupported_certificate
- certificate_revoked

- certificate_expired
- certificate_unknown
- illegal_parameter
- unknown_ca
- access_denied
- decode_error
- decrypt_error
- protocol_version
- insufficient_security
- internal_error
- user_canceled
- no_renegotiation
- unsupported_extension
- unknown.

Actions

get

Parameters

`current_time`: *long integer*, a list of TLS error counters

`ref_time`: *long integer*, a list of TLS error counters

`stats`: *long integer*, a list of TLS error counters

`unexpected_message`: *long integer*, a list of TLS error counters

`bad_record_mac`: *long integer*, a list of TLS error counters

`record_overflow`: *long integer*, a list of TLS error counters

`decompression_failure`: *long integer*, a list of TLS error counters

`handshake_failure`: *long integer*, a list of TLS error counters

`bad_certificate`: *long integer*, a list of TLS error counters

`unsupported_certificate`: *long integer*, a list of TLS error counters

`certificate_revoked`: *long integer*, a list of TLS error counters

`certificate_expired`: *long integer*, a list of TLS error counters

`certificate_unknown`: *long integer*, a list of TLS error counters

`illegal_parameter`: *long integer*, a list of TLS error counters

`unknown_ca`: *long integer*, a list of TLS error counters

`access_denied`: *long integer*, a list of TLS error counters

`decode_error`: *long integer*, a list of TLS error counters

`decrypt_error`: *long integer*, a list of TLS error counters

`protocol_version`: *long integer*, a list of TLS error counters

`insufficient_security`: *long integer*, a list of TLS error counters

`internal_error`: *long integer*, a list of TLS error counters

`user_canceled`: *long integer*, a list of TLS error counters

`no_renegotiation`: *long integer*, a list of TLS error counters

`unsupported_extension`: *long integer*, a list of TLS error counters

`unknown`: *long integer*, a list of TLS error counters

Example

```
# get
input:  0/1 P4G_TLS_ALERT_FATAL_COUNTERS [0] ?
output: 0/1 P4G_TLS_ALERT_FATAL_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_TLS_ALERT_WARNING_COUNTERS

```
# get
<module-index>/<port-index> P4G_TLS_ALERT_WARNING_COUNTERS [<group_
↪index>] ?
```

Description

Returns a list of TLS warning counters. The counters returned corresponds the following TLS warnings:

- close_notify
- unexpected_message
- bad_record_mac

- record_overflow
- decompression_failure
- handshake_failure
- bad_certificate
- unsupported_certificate
- certificate_revoked
- certificate_expired
- certificate_unknown
- illegal_parameter
- unknown_ca
- access_denied
- decode_error
- decrypt_error
- protocol_version
- insufficient_security
- internal_error
- user_canceled
- no_renegotiation
- unsupported_extension
- unknown.

Actions

get

Parameters

`current_time`: *long integer*, a list of TLS warning counters

`ref_time`: *long integer*, a list of TLS warning counters

`close_notify`: *long integer*, a list of TLS warning counters

`unexpected_message`: *long integer*, a list of TLS warning counters

`bad_record_mac`: *long integer*, a list of TLS warning counters

`record_overflow`: *long integer*, a list of TLS warning counters

`decompression_failure`: *long integer*, a list of TLS warning counters

handshake_failure: *long integer*, a list of TLS warning counters

bad_certificate: *long integer*, a list of TLS warning counters

unsupported_certificate: *long integer*, a list of TLS warning counters

certificate_revoked: *long integer*, a list of TLS warning counters

certificate_expired: *long integer*, a list of TLS warning counters

certificate_unknown: *long integer*, a list of TLS warning counters

illegal_parameter: *long integer*, a list of TLS warning counters

unknown_ca: *long integer*, a list of TLS warning counters

access_denied: *long integer*, a list of TLS warning counters

decode_error: *long integer*, a list of TLS warning counters

decrypt_error: *long integer*, a list of TLS warning counters

protocol_version: *long integer*, a list of TLS warning counters

insufficient_security: *long integer*, a list of TLS warning counters

internal_error: *long integer*, a list of TLS warning counters

user_canceled: *long integer*, a list of TLS warning counters

no_renegotiation: *long integer*, a list of TLS warning counters

unsupported_extension: *long integer*, a list of TLS warning counters

unknown: *long integer*, a list of TLS warning counters

Example

```
# get
input:  0/1 P4G_TLS_ALERT_WARNING_COUNTERS [0] ?
output: 0/1 P4G_TLS_ALERT_WARNING_COUNTERS [0] 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123
```

P4G_TLS_CERTIFICATE_FILENAME

```
# set
<module-index>/<port-index> P4G_TLS_CERTIFICATE_FILENAME [<group_index>
↪] <filename>
```

Description

Configure the TLS certificate.

Actions

set

Parameters

`filename`: *string*, the filename of the certificate relative to the FTP TLS folder on the tester.

Example

```
# set
input:  0/1 P4G_TLS_CERTIFICATE_FILENAME [0] "file.txt"
output: <OK>
```

P4G_TLS_CIPHER_SUITES

```
# set
<module-index>/<port-index> P4G_TLS_CIPHER_SUITES [<group_index>]
↪<ciphers>

# get
<module-index>/<port-index> P4G_TLS_CIPHER_SUITES [<group_index>] ?
```

Description

Configure the list of ciphers to announce in order of priorities.

Actions

set, get

Parameters

`ciphers`: *hex list*, sequence of ciphers identified by theirs IANA number in order of priority.

Example

```
# set
input:  0/1 P4G_TLS_CIPHER_SUITES [0] 0x000200030004
output: <OK>

# get
input:  0/1 P4G_TLS_CIPHER_SUITES [0] ?
output: 0/1 P4G_TLS_CIPHER_SUITES [0] 0x000200030004
```

P4G_TLS_CLOSE_NOTIFY

```
# set
<module-index>/<port-index> P4G_TLS_CLOSE_NOTIFY [<group_index>] <on_
↪off>

# get
<module-index>/<port-index> P4G_TLS_CLOSE_NOTIFY [<group_index>] ?
```

Description

Enable/Disable TLS sending close notify alert on connection tear-down.

Actions

set, get

Parameters

on_off: *byte*, whether TLS sends close notify alert on connection tear-down.

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_TLS_CLOSE_NOTIFY [0] NO
output: <OK>

# get
input:  0/1 P4G_TLS_CLOSE_NOTIFY [0] ?
output: 0/1 P4G_TLS_CLOSE_NOTIFY [0] NO
```

P4G_TLS_DHPARAMS_FILENAME

```
# set
<module-index>/<port-index> P4G_TLS_DHPARAMS_FILENAME [<group_index>]
↪<filename>
```

Description

Configure TLS DH parameters, if not set a default set will be used.

Actions

set

Parameters

`filename`: *string*, the filename of the TLS DH parameters relative to the FTP TLS folder on the tester.

Example

```
# set
input:  0/1 P4G_TLS_DHPARAMS_FILENAME [0] "file.txt"
output: <OK>
```

P4G_TLS_ENABLE

```
# set
<module-index>/<port-index> P4G_TLS_ENABLE [<group_index>] <on_off>

# get
<module-index>/<port-index> P4G_TLS_ENABLE [<group_index>] ?
```

Description

Enable/Disable TLS.

Actions

set, get

Parameters

`on_off`: *byte*, specifying whether to enable TLS

- `NO = 0`
- `YES = 1`

Example

```
# set
input:  0/1 P4G_TLS_ENABLE [0] NO
output: <OK>

# get
input:  0/1 P4G_TLS_ENABLE [0] ?
output: 0/1 P4G_TLS_ENABLE [0] NO
```

P4G_TLS_HANDSHAKE_HIST

```
# get
<module-index>/<port-index> P4G_TLS_HANDSHAKE_HIST [<group_index>] ?
```

Description

Returns a histogram over TLS connection handshake times, with start and interval values as configured by P4G_PAYLOAD_HIST_CONF.

Actions

get

Parameters

connection_count: *integer*, a histogram over TLS connection handshake times

min_connection_handshake_time: *long integer*, a histogram over TLS connection handshake times

max_connection_handshake_time: *long integer*, a histogram over TLS connection handshake times

avg_connection_handshake_time: *long integer*, a histogram over TLS connection handshake times

start: *long integer*, a histogram over TLS connection handshake times

interval: *long integer*, a histogram over TLS connection handshake times

bin_00: *integer*, a histogram over TLS connection handshake times

bin_01: *integer*, a histogram over TLS connection handshake times

bin_02: *integer*, a histogram over TLS connection handshake times

bin_03: *integer*, a histogram over TLS connection handshake times

bin_04: *integer*, a histogram over TLS connection handshake times

bin_05: *integer*, a histogram over TLS connection handshake times

bin_06: *integer*, a histogram over TLS connection handshake times

bin_07: *integer*, a histogram over TLS connection handshake times

bin_08: *integer*, a histogram over TLS connection handshake times

bin_09: *integer*, a histogram over TLS connection handshake times

bin_10: *integer*, a histogram over TLS connection handshake times

bin_11: *integer*, a histogram over TLS connection handshake times

bin_12: *integer*, a histogram over TLS connection handshake times

bin_13: *integer*, a histogram over TLS connection handshake times

bin_14: *integer*, a histogram over TLS connection handshake times

bin_15: *integer*, a histogram over TLS connection handshake times

bin_16: *integer*, a histogram over TLS connection handshake times

bin_17: *integer*, a histogram over TLS connection handshake times

bin_18: *integer*, a histogram over TLS connection handshake times

bin_19: *integer*, a histogram over TLS connection handshake times

bin_20: *integer*, a histogram over TLS connection handshake times

bin_21: *integer*, a histogram over TLS connection handshake times

bin_22: *integer*, a histogram over TLS connection handshake times

bin_23: *integer*, a histogram over TLS connection handshake times

bin_24: *integer*, a histogram over TLS connection handshake times

bin_25: *integer*, a histogram over TLS connection handshake times

bin_26: *integer*, a histogram over TLS connection handshake times

bin_27: *integer*, a histogram over TLS connection handshake times

bin_28: *integer*, a histogram over TLS connection handshake times

bin_29: *integer*, a histogram over TLS connection handshake times

bin_30: *integer*, a histogram over TLS connection handshake times

bin_31: *integer*, a histogram over TLS connection handshake times

Example

```
# get
input:  0/1 P4G_TLS_HANDSHAKE_HIST [0] ?
output: 0/1 P4G_TLS_HANDSHAKE_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TLS_MAX_RECORD_SIZE

```
# set
<module-index>/<port-index> P4G_TLS_MAX_RECORD_SIZE [<group_index>]
↪<size>

# get
<module-index>/<port-index> P4G_TLS_MAX_RECORD_SIZE [<group_index>] ?
```

Description

Configure the maximum outgoing TLS record size.

Actions

set, get

Parameters

`size`: *integer*, maximum outgoing record size in the interval (0, 16384], default value 8087.

Example

```
# set
input:  0/1 P4G_TLS_MAX_RECORD_SIZE [0] 1
output: <OK>

# get
input:  0/1 P4G_TLS_MAX_RECORD_SIZE [0] ?
output: 0/1 P4G_TLS_MAX_RECORD_SIZE [0] 1
```

P4G_TLS_MIN_REQ_PROTOCOL_VER

```
# get
<module-index>/<port-index> P4G_TLS_MIN_REQ_PROTOCOL_VER [<group_index>
↪] ?
```

Description

Returns the minimum TLS protocol version required by the configured list of cipher suites. Each cipher suite has a minimum required TLS protocol version that will support the cipher suite. The minimum required TLS protocol version for a list of cipher suites is the lowest minimum required TLS protocol version of all the cipher suites in the list.

Actions

get

Parameters

tls_version: *byte*, the minimum TLS protocol version required by the configured list of cipher suites.

- SSLV3 = 0
- TLS10 = 1
- TLS11 = 2
- TLS12 = 3

Example

```
# get
input:  0/1 P4G_TLS_MIN_REQ_PROTOCOL_VER [0] ?
output: 0/1 P4G_TLS_MIN_REQ_PROTOCOL_VER [0] SSLV3
```

P4G_TLS_PRIVATE_KEY_FILENAME

```
# set
<module-index>/<port-index> P4G_TLS_PRIVATE_KEY_FILENAME [<group_index>
↪] <filename>
```

Description

Configure the private key matching the TLS certificate.

Actions

set

Parameters

`filename`: *string*, the filename of the private key relative to the FTP TLS folder on the tester.

Example

```
# set
input:  0/1 P4G_TLS_PRIVATE_KEY_FILENAME [0] "file.txt"
output: <OK>
```

P4G_TLS_PROTOCOL_VER

```
# set
<module-index>/<port-index> P4G_TLS_PROTOCOL_VER [<group_index>] <tls_
↪version>

# get
<module-index>/<port-index> P4G_TLS_PROTOCOL_VER [<group_index>] ?
```

Description

Configures the desired TLS protocol version. More specifically the TLS version configured is the protocol version advertised by the client in the Client Hello message, and the highest TLS protocol version accepted by the server. If the protocol_version in the Client Hello message is higher than the highest protocol version accepted by the server, the TLS Handshake will fail.

Actions

set, get

Parameters

tls_version: ***byte*****, the highest supported TLS protocol version**

- SSLV3 = 0
- TLS10 = 1
- TLS11 = 2
- TLS12 = 3

Example

```
# set
input:  0/1 P4G_TLS_PROTOCOL_VER [0] SSLV3
output: <OK>

# get
input:  0/1 P4G_TLS_PROTOCOL_VER [0] ?
output: 0/1 P4G_TLS_PROTOCOL_VER [0] SSLV3
```

P4G_TLS_RX_PAYLOAD_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TLS_RX_PAYLOAD_BYTES_HIST [<group_
↪index>] ?
```

Description

Returns a histogram over number of TLS Payload bytes received, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of TLS Payload bytes received

`min_byte_count`: *long integer*, a histogram over number of TLS Payload bytes received

`max_byte_count`: *long integer*, a histogram over number of TLS Payload bytes received

`avg_byte_count`: *long integer*, a histogram over number of TLS Payload bytes received

`start`: *long integer*, a histogram over number of TLS Payload bytes received

`interval`: *long integer*, a histogram over number of TLS Payload bytes received

`bin_00`: *integer*, a histogram over number of TLS Payload bytes received

`bin_01`: *integer*, a histogram over number of TLS Payload bytes received

`bin_02`: *integer*, a histogram over number of TLS Payload bytes received

`bin_03`: *integer*, a histogram over number of TLS Payload bytes received

`bin_04`: *integer*, a histogram over number of TLS Payload bytes received

`bin_05`: *integer*, a histogram over number of TLS Payload bytes received

`bin_06`: *integer*, a histogram over number of TLS Payload bytes received

`bin_07`: *integer*, a histogram over number of TLS Payload bytes received

`bin_08`: *integer*, a histogram over number of TLS Payload bytes received

`bin_09`: *integer*, a histogram over number of TLS Payload bytes received

`bin_10`: *integer*, a histogram over number of TLS Payload bytes received

`bin_11`: *integer*, a histogram over number of TLS Payload bytes received

`bin_12`: *integer*, a histogram over number of TLS Payload bytes received

`bin_13`: *integer*, a histogram over number of TLS Payload bytes received

`bin_14`: *integer*, a histogram over number of TLS Payload bytes received

`bin_15`: *integer*, a histogram over number of TLS Payload bytes received

`bin_16`: *integer*, a histogram over number of TLS Payload bytes received

bin_17: *integer*, a histogram over number of TLS Payload bytes received

bin_18: *integer*, a histogram over number of TLS Payload bytes received

bin_19: *integer*, a histogram over number of TLS Payload bytes received

bin_20: *integer*, a histogram over number of TLS Payload bytes received

bin_21: *integer*, a histogram over number of TLS Payload bytes received

bin_22: *integer*, a histogram over number of TLS Payload bytes received

bin_23: *integer*, a histogram over number of TLS Payload bytes received

bin_24: *integer*, a histogram over number of TLS Payload bytes received

bin_25: *integer*, a histogram over number of TLS Payload bytes received

bin_26: *integer*, a histogram over number of TLS Payload bytes received

bin_27: *integer*, a histogram over number of TLS Payload bytes received

bin_28: *integer*, a histogram over number of TLS Payload bytes received

bin_29: *integer*, a histogram over number of TLS Payload bytes received

bin_30: *integer*, a histogram over number of TLS Payload bytes received

bin_31: *integer*, a histogram over number of TLS Payload bytes received

Example

```
# get
input:  0/1 P4G_TLS_RX_PAYLOAD_BYTES_HIST [0] ?
output: 0/1 P4G_TLS_RX_PAYLOAD_BYTES_HIST [0] 1 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TLS_RX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_TLS_RX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the TLS Rx payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TLS Rx payload counters.

`ref_time`: *long integer*, a list of the TLS Rx payload counters.

`byte_count`: *long integer*, a list of the TLS Rx payload counters.

`byte_per_second`: *long integer*, a list of the TLS Rx payload counters.

Example

```
# get
input:  0/1 P4G_TLS_RX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_TLS_RX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_TLS_SERVER_NAME

```
# set
<module-index>/<port-index> P4G_TLS_SERVER_NAME [<group_index>]
↪<server_name>

# get
<module-index>/<port-index> P4G_TLS_SERVER_NAME [<group_index>] ?
```

Description

Configure the server name advertised by the client in the TLS SNI (Server Name Indication) extension. Both the client and server must be configured with the same server_name, as the server will check the server name in Client Hello message. If server name is not configured (or configured blank), the SNI extension will not be inserted in the Client Hello message.

Actions

set, get

Parameters

`server_name`: *string*, server name inserted in the SNI TLS extension

Example

```
# set
input:  0/1 P4G_TLS_SERVER_NAME [0] "www.myservername.com"
output: <OK>

# get
input:  0/1 P4G_TLS_SERVER_NAME [0] ?
output: 0/1 P4G_TLS_SERVER_NAME [0] "www.myservername.com"
```

P4G_TLS_STATE_CURRENT

```
# get
<module-index>/<port-index> P4G_TLS_STATE_CURRENT [<group_index>] ?
```

Description

Returns a list of the current TLS state counters. The counters returned corresponds the the following TLS states:

- TLS_INACTIVE
- TLS_HANDSHAKING
- TLS_HANDSHAKE_DONE
- TLS_HANDSHAKE_FAILED
- TLS_FAILED
- TLS_INTERNAL_FAILED
- TLS_CLOSE_NOTIFY
- TLS_DONE

Actions

get

Parameters

`current_time`: *long integer*, a list of the current TLS state counters

`ref_time`: *long integer*, a list of the current TLS state counters

`tls_inactive`: *long integer*, a list of the current TLS state counters

`tls_handshaking`: *long integer*, a list of the current TLS state counters

`tls_handshake_done`: *long integer*, a list of the current TLS state counters

`tls_handshake_failed`: *long integer*, a list of the current TLS state counters

`tls_failed`: *long integer*, a list of the current TLS state counters

`tls_internal_failed`: *long integer*, a list of the current TLS state counters

`tls_close_notify`: *long integer*, a list of the current TLS state counters

`tls_done`: *long integer*, a list of the current TLS state counters

Example

```
# get
input:  0/1 P4G_TLS_STATE_CURRENT [0] ?
output: 0/1 P4G_TLS_STATE_CURRENT [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_TLS_STATE_RATE

```
# get
<module-index>/<port-index> P4G_TLS_STATE_RATE [<group_index>] ?
```

Description

Returns a list of the TLS state rates measured in per second. The counters returned corresponds the the following TLS states:

- TLS_INACTIVE
- TLS_HANDSHAKING
- TLS_HANDSHAKE_DONE
- TLS_HANDSHAKE_FAILED
- TLS_FAILED
- TLS_INTERNAL_FAILED
- TLS_CLOSE_NOTIFY
- TLS_DONE

Actions

get

Parameters

`current_time`: *long integer*, a list of the TLS state rates measured in per second

`ref_time`: *long integer*, a list of the TLS state rates measured in per second

`tls_inactive`: *long integer*, a list of the TLS state rates measured in per second

`tls_handshaking`: *long integer*, a list of the TLS state rates measured in per second

`tls_handshake_done`: *long integer*, a list of the TLS state rates measured in per second

`tls_handshake_failed`: *long integer*, a list of the TLS state rates measured in per second

`tls_failed`: *long integer*, a list of the TLS state rates measured in per second

`tls_internal_failed`: *long integer*, a list of the TLS state rates measured in per second

`tls_close_notify`: *long integer*, a list of the TLS state rates measured in per second

`tls_done`: *long integer*, a list of the TLS state rates measured in per second

Example

```
# get
input:  0/1 P4G_TLS_STATE_RATE [0] ?
output: 0/1 P4G_TLS_STATE_RATE [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_TLS_STATE_TOTAL

```
# get
<module-index>/<port-index> P4G_TLS_STATE_TOTAL [<group_index>] ?
```

Description

Returns a list of the total TLS state counters. The counters returned corresponds the the following TLS states:

- TLS_INACTIVE
- TLS_HANDSHAKING
- TLS_HANDSHAKE_DONE
- TLS_HANDSHAKE_FAILED
- TLS_FAILED
- TLS_INTERNAL_FAILED
- TLS_CLOSE_NOTIFY
- TLS_DONE

Actions

get

Parameters

`current_time`: *long integer*, a list of the total TLS state counters

`ref_time`: *long integer*, a list of the total TLS state counters

`tls_inactive`: *long integer*, a list of the total TLS state counters

`tls_handshaking`: *long integer*, a list of the total TLS state counters

`tls_handshake_done`: *long integer*, a list of the total TLS state counters

`tls_handshake_failed`: *long integer*, a list of the total TLS state counters

`tls_failed`: *long integer*, a list of the total TLS state counters

`tls_internal_failed`: *long integer*, a list of the total TLS state counters

`tls_close_notify`: *long integer*, a list of the total TLS state counters

`tls_done`: *long integer*, a list of the total TLS state counters

Example

```
# get
input:  0/1 P4G_TLS_STATE_TOTAL [0] ?
output: 0/1 P4G_TLS_STATE_TOTAL [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_TLS_TX_PAYLOAD_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_TLS_TX_PAYLOAD_BYTES_HIST [<group_
↪index>] ?
```

Description

Returns a histogram over number of TLS Payload bytes transmitted, with start and interval values as configured by `P4G_PAYLOAD_HIST_CONF`.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of TLS Payload bytes transmitted

`min_byte_count`: *long integer*, a histogram over number of TLS Payload bytes transmitted

`max_byte_count`: *long integer*, a histogram over number of TLS Payload bytes transmitted

`avg_byte_count`: *long integer*, a histogram over number of TLS Payload bytes transmitted

`start`: *long integer*, a histogram over number of TLS Payload bytes transmitted

`interval`: *long integer*, a histogram over number of TLS Payload bytes transmitted

`bin_00`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_01`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_02`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_03`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_04`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_05`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_06`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_07`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_08`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_09`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_10`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_11`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_12`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_13`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_14`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_15`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_16`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_17`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_18`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_19`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_20`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_21`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_22`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_23`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_24`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_25`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_26`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_27`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_28`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_29`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_30`: *integer*, a histogram over number of TLS Payload bytes transmitted

`bin_31`: *integer*, a histogram over number of TLS Payload bytes transmitted

Example

```
# get
input:  0/1 P4G_TLS_TX_PAYLOAD_BYTES_HIST [0] ?
output: 0/1 P4G_TLS_TX_PAYLOAD_BYTES_HIST [0] 1 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_TLS_TX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_TLS_TX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the TLS Tx payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the TLS Tx payload counters.

`ref_time`: *long integer*, a list of the TLS Tx payload counters.

`byte_count`: *long integer*, a list of the TLS Tx payload counters.

`byte_per_second`: *long integer*, a list of the TLS Tx payload counters.

Example

```
# get
input:  0/1 P4G_TLS_TX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_TLS_TX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_TRANSACTION_HIST_CONF

```
# set
<module-index>/<port-index> P4G_TRANSACTION_HIST_CONF [<group_index>]
 ↪<start> <interval>

# get
<module-index>/<port-index> P4G_TRANSACTION_HIST_CONF [<group_index>] ?
```

Description

Sets the start value and the interval size for the transaction histogram (P4G_APP_TRANSACTION_HIST).

Actions

set, get

Parameters

`start`: *long integer*, tart value of first histogram interval

`interval`: *long integer*, histogram interval size

Example

```
# set
input:  0/1 P4G_TRANSACTION_HIST_CONF [0] 1 1
output: <OK>

# get
input:  0/1 P4G_TRANSACTION_HIST_CONF [0] ?
output: 0/1 P4G_TRANSACTION_HIST_CONF [0] 1 1
```

P4G_UDP_PACKET_SIZE_MINMAX

```
# set
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_MINMAX [<group_index>]
 ↪<size_min> <size_max>

# get
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_MINMAX [<group_index>]␣
 ↪?
```

Description

Configure the minimum and maximum values of the range for UDP packet size. Both values are included in the range. Relevant when P4G_UDP_PACKET_SIZE_TYPE is set to INCREMENT or RANDOM.

Actions

set, get

Parameters

`size_min`: *integer*, the minimum value of UDP packet size

`size_max`: *integer*, the maximum value of UDP packet size

Example

```
# set
input:  0/1 P4G_UDP_PACKET_SIZE_MINMAX [0] 1 1
output: <OK>

# get
input:  0/1 P4G_UDP_PACKET_SIZE_MINMAX [0] ?
output: 0/1 P4G_UDP_PACKET_SIZE_MINMAX [0] 1 1
```

P4G_UDP_PACKET_SIZE_TYPE

```
# set
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_TYPE [<group_index>]
↪<packet_size_type>

# get
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_TYPE [<group_index>] ?
```

Description

Specifies the UDP packet size type for a CG. The packet size can either be fixed size identical for all connections in the CG, incrementing or random. The individual packet size for a specific connection is always constant once the incrementing or random value has been created. Refer to `P4G_UDP_PACKET_SIZE_MINMAX` command for information on how to configure min and max values.

Actions

set, get

Parameters

`packet_size_type`: *byte*, specifying how UDP packet size is set

- `FIXED = 0`
- `INCREMENT = 1`
- `RANDOM = 2`

Example

```
# set
input:  0/1 P4G_UDP_PACKET_SIZE_TYPE [0] FIXED
output: <OK>

# get
input:  0/1 P4G_UDP_PACKET_SIZE_TYPE [0] ?
output: 0/1 P4G_UDP_PACKET_SIZE_TYPE [0] FIXED
```

P4G_UDP_PACKET_SIZE_VALUE

```
# set
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_VALUE [<group_index>]
 ↪<size>

# get
<module-index>/<port-index> P4G_UDP_PACKET_SIZE_VALUE [<group_index>] ?
```

Description

Configure the fixed UDP packet size value. Relevant when P4G_UDP_PACKET_SIZE_TYPE is set to FIXED.

Actions

set, get

Parameters

`size`: *integer*, the fixed value of UDP packet size

Example

```
# set
input:  0/1 P4G_UDP_PACKET_SIZE_VALUE [0] 1
output: <OK>

# get
input:  0/1 P4G_UDP_PACKET_SIZE_VALUE [0] ?
output: 0/1 P4G_UDP_PACKET_SIZE_VALUE [0] 1
```

P4G_UDP_RX_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_UDP_RX_BYTES_HIST [<group_index>] ?
```

Description

Returns a histogram over number of UDP bytes received, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of UDP bytes received

`min_byte_count`: *long integer*, a histogram over number of UDP bytes received

`max_byte_count`: *long integer*, a histogram over number of UDP bytes received

`avg_byte_count`: *long integer*, a histogram over number of UDP bytes received

`start`: *long integer*, a histogram over number of UDP bytes received

`interval`: *long integer*, a histogram over number of UDP bytes received

`bin_00`: *integer*, a histogram over number of UDP bytes received

`bin_01`: *integer*, a histogram over number of UDP bytes received

`bin_02`: *integer*, a histogram over number of UDP bytes received

`bin_03`: *integer*, a histogram over number of UDP bytes received

`bin_04`: *integer*, a histogram over number of UDP bytes received

`bin_05`: *integer*, a histogram over number of UDP bytes received

`bin_06`: *integer*, a histogram over number of UDP bytes received

`bin_07`: *integer*, a histogram over number of UDP bytes received

`bin_08`: *integer*, a histogram over number of UDP bytes received

`bin_09`: *integer*, a histogram over number of UDP bytes received

`bin_10`: *integer*, a histogram over number of UDP bytes received

`bin_11`: *integer*, a histogram over number of UDP bytes received

`bin_12`: *integer*, a histogram over number of UDP bytes received

`bin_13`: *integer*, a histogram over number of UDP bytes received

`bin_14`: *integer*, a histogram over number of UDP bytes received

`bin_15`: *integer*, a histogram over number of UDP bytes received

`bin_16`: *integer*, a histogram over number of UDP bytes received

bin_17: *integer*, a histogram over number of UDP bytes received

bin_18: *integer*, a histogram over number of UDP bytes received

bin_19: *integer*, a histogram over number of UDP bytes received

bin_20: *integer*, a histogram over number of UDP bytes received

bin_21: *integer*, a histogram over number of UDP bytes received

bin_22: *integer*, a histogram over number of UDP bytes received

bin_23: *integer*, a histogram over number of UDP bytes received

bin_24: *integer*, a histogram over number of UDP bytes received

bin_25: *integer*, a histogram over number of UDP bytes received

bin_26: *integer*, a histogram over number of UDP bytes received

bin_27: *integer*, a histogram over number of UDP bytes received

bin_28: *integer*, a histogram over number of UDP bytes received

bin_29: *integer*, a histogram over number of UDP bytes received

bin_30: *integer*, a histogram over number of UDP bytes received

bin_31: *integer*, a histogram over number of UDP bytes received

Example

```
# get
input:  0/1 P4G_UDP_RX_BYTES_HIST [0] ?
output: 0/1 P4G_UDP_RX_BYTES_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_UDP_RX_PACKET_COUNTERS

```
# get
<module-index>/<port-index> P4G_UDP_RX_PACKET_COUNTERS [<group_index>]␣
↪?
```

Description

Returns a list of the UDP RX packet counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the UDP RX packet counters.

`ref_time`: *long integer*, a list of the UDP RX packet counters.

`packet_count`: *long integer*, a list of the UDP RX packet counters.

`packet_per_second`: *long integer*, a list of the UDP RX packet counters.

Example

```
# get
input:  0/1 P4G_UDP_RX_PACKET_COUNTERS [0] ?
output: 0/1 P4G_UDP_RX_PACKET_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_UDP_RX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_UDP_RX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the UDP RX payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the UDP RX payload counters.

`ref_time`: *long integer*, a list of the UDP RX payload counters.

`byte_count`: *long integer*, a list of the UDP RX payload counters.

`byte_per_second`: *long integer*, a list of the UDP RX payload counters.

Example

```
# get
input:  0/1 P4G_UDP_RX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_UDP_RX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_UDP_STATE_CURRENT

```
# get
<module-index>/<port-index> P4G_UDP_STATE_CURRENT [<group_index>] ?
```

Description

Returns a list of the current UDP state counters. The counters returned corresponds the the following UDP states:

- CLOSED: The connection structure has been created, but has not been 'ramped up' yet.
- OPEN: The connection has been 'ramped up', and is ready to transmit or receive data.
- ACTIVE: The connection is actively transmitting data.

Actions

get

Parameters

`current_time`: *long integer*, a list of the current UDP state counters

`ref_time`: *long integer*, a list of the current UDP state counters

`closed`: *long integer*, a list of the current UDP state counters

`opened`: *long integer*, a list of the current UDP state counters

`active`: *long integer*, a list of the current UDP state counters

Example

```
# get
input:  0/1 P4G_UDP_STATE_CURRENT [0] ?
output: 0/1 P4G_UDP_STATE_CURRENT [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_UDP_STATE_RATE

```
# get
<module-index>/<port-index> P4G_UDP_STATE_RATE [<group_index>] ?
```

Description

Returns a list of the UDP state rates measured in connections/second. The counters returned corresponds the the following UDP state rates:

- CLOSED: The connection structure has been created, but has not been 'ramped up' yet.
- OPEN: The connection has been 'ramped up', and is ready to transmit or receive data.
- ACTIVE: The connection is actively transmitting data.

Actions

get

Parameters

`current_time`: *long integer*, a list of the UDP state rates measured in connections/second.

`ref_time`: *long integer*, a list of the UDP state rates measured in connections/second.

`closed`: *long integer*, a list of the UDP state rates measured in connections/second.

`open`: *long integer*, a list of the UDP state rates measured in connections/second.

`active`: *long integer*, a list of the UDP state rates measured in connections/second.

Example

```
# get
input:  0/1 P4G_UDP_STATE_RATE [0] ?
output: 0/1 P4G_UDP_STATE_RATE [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_UDP_STATE_TOTAL

```
# get
<module-index>/<port-index> P4G_UDP_STATE_TOTAL [<group_index>] ?
```

Description

Returns a list of the total UDP state counters. The counters returned corresponds the the following UDP states:

- CLOSED: The connection structure has been created, but has not been 'ramped up' yet.
- OPEN: The connection has been 'ramped up', and is ready to transmit or receive data.
- ACTIVE: The connection is actively transmitting data.

Actions

get

Parameters

`current_time`: *long integer*, a list of the total UDP state counters.

`ref_time`: *long integer*, a list of the total UDP state counters.

`closed`: *long integer*, a list of the total UDP state counters.

`opened`: *long integer*, a list of the total UDP state counters.

`active`: *long integer*, a list of the total UDP state counters.

Example

```
# get
input:  0/1 P4G_UDP_STATE_TOTAL [0] ?
output: 0/1 P4G_UDP_STATE_TOTAL [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123
```

P4G_UDP_TX_BYTES_HIST

```
# get
<module-index>/<port-index> P4G_UDP_TX_BYTES_HIST [<group_index>] ?
```

Description

Returns a histogram over number of UDP bytes transmitted, with start and interval values as configured by *P4G_PAYLOAD_HIST_CONF*.

Actions

get

Parameters

`connection_count`: *integer*, a histogram over number of UDP bytes transmitted

`min_byte_count`: *long integer*, a histogram over number of UDP bytes transmitted

`max_byte_count`: *long integer*, a histogram over number of UDP bytes transmitted

`avg_byte_count`: *long integer*, a histogram over number of UDP bytes transmitted

`start`: *long integer*, a histogram over number of UDP bytes transmitted

`interval`: *long integer*, a histogram over number of UDP bytes transmitted

`bin_00`: *integer*, a histogram over number of UDP bytes transmitted

`bin_01`: *integer*, a histogram over number of UDP bytes transmitted

`bin_02`: *integer*, a histogram over number of UDP bytes transmitted

`bin_03`: *integer*, a histogram over number of UDP bytes transmitted

`bin_04`: *integer*, a histogram over number of UDP bytes transmitted

`bin_05`: *integer*, a histogram over number of UDP bytes transmitted

`bin_06`: *integer*, a histogram over number of UDP bytes transmitted

`bin_07`: *integer*, a histogram over number of UDP bytes transmitted

`bin_08`: *integer*, a histogram over number of UDP bytes transmitted

`bin_09`: *integer*, a histogram over number of UDP bytes transmitted

`bin_10`: *integer*, a histogram over number of UDP bytes transmitted

`bin_11`: *integer*, a histogram over number of UDP bytes transmitted

`bin_12`: *integer*, a histogram over number of UDP bytes transmitted

`bin_13`: *integer*, a histogram over number of UDP bytes transmitted

`bin_14`: *integer*, a histogram over number of UDP bytes transmitted

`bin_15`: *integer*, a histogram over number of UDP bytes transmitted

`bin_16`: *integer*, a histogram over number of UDP bytes transmitted

`bin_17`: *integer*, a histogram over number of UDP bytes transmitted

`bin_18`: *integer*, a histogram over number of UDP bytes transmitted

`bin_19`: *integer*, a histogram over number of UDP bytes transmitted

`bin_20`: *integer*, a histogram over number of UDP bytes transmitted

`bin_21`: *integer*, a histogram over number of UDP bytes transmitted

`bin_22`: *integer*, a histogram over number of UDP bytes transmitted

`bin_23`: *integer*, a histogram over number of UDP bytes transmitted

`bin_24`: *integer*, a histogram over number of UDP bytes transmitted

bin_25: *integer*, a histogram over number of UDP bytes transmitted

bin_26: *integer*, a histogram over number of UDP bytes transmitted

bin_27: *integer*, a histogram over number of UDP bytes transmitted

bin_28: *integer*, a histogram over number of UDP bytes transmitted

bin_29: *integer*, a histogram over number of UDP bytes transmitted

bin_30: *integer*, a histogram over number of UDP bytes transmitted

bin_31: *integer*, a histogram over number of UDP bytes transmitted

Example

```
# get
input:  0/1 P4G_UDP_TX_BYTES_HIST [0] ?
output: 0/1 P4G_UDP_TX_BYTES_HIST [0] 1 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 1 1 1 1 1 1 1 1 1 1 1 1 1 1␣
↪1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

P4G_UDP_TX_PACKET_COUNTERS

```
# get
<module-index>/<port-index> P4G_UDP_TX_PACKET_COUNTERS [<group_index>]␣
↪?
```

Description

Returns a list of the UDP TX packet counters.

Actions

get

Parameters

current_time: *long integer*, a list of the UDP TX packet counters.

ref_time: *long integer*, a list of the UDP TX packet counters.

packet_count: *long integer*, a list of the UDP TX packet counters.

packet_per_second: *long integer*, a list of the UDP TX packet counters.

Example

```
# get
input:  0/1 P4G_UDP_TX_PACKET_COUNTERS [0] ?
output: 0/1 P4G_UDP_TX_PACKET_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_UDP_TX_PAYLOAD_COUNTERS

```
# get
<module-index>/<port-index> P4G_UDP_TX_PAYLOAD_COUNTERS [<group_index>
↪] ?
```

Description

Returns a list of the UDP TX payload counters.

Actions

get

Parameters

`current_time`: *long integer*, a list of the UDP TX payload counters.

`ref_time`: *long integer*, a list of the UDP TX payload counters.

`byte_count`: *long integer*, a list of the UDP TX payload counters.

`byte_per_second`: *long integer*, a list of the UDP TX payload counters.

Example

```
# get
input:  0/1 P4G_UDP_TX_PAYLOAD_COUNTERS [0] ?
output: 0/1 P4G_UDP_TX_PAYLOAD_COUNTERS [0] 123456789123 123456789123␣
↪123456789123 123456789123
```

P4G_USER_STATE_CURRENT

```
# get
<module-index>/<port-index> P4G_USER_STATE_CURRENT [<group_index>] ?
```

Description

Returns a list of the current user state counters. A user is identified by a Client IP address. The counters returned corresponds the the following user states:

- INIT: The user has been created, but has no open connections yet.
- ACTIVE: The user has at least one open connection.
- SUCCESS: The user has successfully transmitted and received all payload.
- FAILED: The user has failed in transmitting or receiving all payload. STOPPED The user has been stopped due to ramp-down.
- INACTIVE: All the users connection is closed, but the user has not been destroyed yet.

Actions

get

Parameters

`current_time`: *long integer*, a list of the current user state counters.

`ref_time`: *long integer*, a list of the current user state counters.

`init`: *long integer*, a list of the current user state counters.

`active`: *long integer*, a list of the current user state counters.

`success`: *long integer*, a list of the current user state counters.

`failed`: *long integer*, a list of the current user state counters.

`stopped`: *long integer*, a list of the current user state counters.

`inactive`: *long integer*, a list of the current user state counters.

Example

```
# get
input:  0/1 P4G_USER_STATE_CURRENT [0] ?
output: 0/1 P4G_USER_STATE_CURRENT [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_USER_STATE_RATE

```
# get
<module-index>/<port-index> P4G_USER_STATE_RATE [<group_index>] ?
```

Description

Returns a list of the user state rates measured in users/second. A user is identified by a Client IP address. The counters returned corresponds the the following user states:

- INIT: The user has been created, but has no open connections yet.
- ACTIVE: The user has at least one open connection.
- SUCCESS: The user has successfully transmitted and received all payload.
- FAILED: The user has failed in transmitting or receiving all payload. STOPPED The user has been stopped due to ramp-down.
- INACTIVE: All the users connection is closed, but the user has not been destroyed yet.

Actions

get

Parameters

`current_time`: *long integer*, a list of the user state rates measured in users/second.

`ref_time`: *long integer*, a list of the user state rates measured in users/second.

`stats`: *long integer*, a list of the user state rates measured in users/second.

`init`: *long integer*, a list of the user state rates measured in users/second.

`active`: *long integer*, a list of the user state rates measured in users/second.

`success`: *long integer*, a list of the user state rates measured in users/second.

`failed`: *long integer*, a list of the user state rates measured in users/second.

`stopped`: *long integer*, a list of the user state rates measured in users/second.

Example

```
# get
input:  0/1 P4G_USER_STATE_RATE [0] ?
output: 0/1 P4G_USER_STATE_RATE [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_USER_STATE_TOTAL

```
# get
<module-index>/<port-index> P4G_USER_STATE_TOTAL [<group_index>] ?
```

Description

Returns a list of the total user state counters. A user is identified by a Client IP address. The counters returned corresponds the the following user states:

- INIT: The user has been created, but has no open connections yet.
- ACTIVE: The user has at least one open connection.
- SUCCESS: The user has successfully transmitted and received all payload.
- FAILED: The user has failed in transmitting or receiving all payload. STOPPED The user has been stopped due to ramp-down.
- INACTIVE: All the users connection is closed, but the user has not been destroyed yet.

Actions

get

Parameters

`current_time`: *long integer*, a list of the total user state counters.

`ref_time`: *long integer*, a list of the total user state counters.

`init`: *long integer*, a list of the total user state counters.

`active`: *long integer*, a list of the total user state counters.

`success`: *long integer*, a list of the total user state counters.

`failed`: *long integer*, a list of the total user state counters.

`stopped`: *long integer*, a list of the total user state counters.

`inactive`: *long integer*, a list of the total user state counters.

Example

```
# get
input:  0/1 P4G_USER_STATE_TOTAL [0] ?
output: 0/1 P4G_USER_STATE_TOTAL [0] 123456789123 123456789123␣
↪123456789123 123456789123 123456789123 123456789123 123456789123␣
↪123456789123
```

P4G_VLAN_ENABLE

```
# set
<module-index>/<port-index> P4G_VLAN_ENABLE [<group_index>] <on_off>

# get
<module-index>/<port-index> P4G_VLAN_ENABLE [<group_index>] ?
```

Description

Specify whether to insert a VLAN tag header upon transmit.

Actions

set, get

Parameters

on_off: *byte*, **specifying whether to enable VLAN tag**

- NO = 0
- YES = 1

Example

```
# set
input:  0/1 P4G_VLAN_ENABLE [0] OFF
output: <OK>

# get
input:  0/1 P4G_VLAN_ENABLE [0] ?
output: 0/1 P4G_VLAN_ENABLE [0] OFF
```

P4G_VLAN_TCI

```
# set
<module-index>/<port-index> P4G_VLAN_TCI [<group_index>] <tci>

# get
<module-index>/<port-index> P4G_VLAN_TCI [<group_index>] ?
```

Description

Specify the VLAN TCI.

Actions

set, get

Parameters

`tci`: *hex2*, specifying the 16 bit TCI

Example

```
# set
input:  0/1 P4G_VLAN_TCI [0] 0x0000
output: <OK>

# get
input:  0/1 P4G_VLAN_TCI [0] ?
output: 0/1 P4G_VLAN_TCI [0] 0x0000
```

CHAPTER

FOUR

GLOSSARY OF TERMS

A
Data format `address`, a dot-separated IPv4 address, e.g. 192.168.1.200.

A*
Data format `address list`, a variable number of *addresses* with spaces in between, e.g. 192.168.1.1 192.168.1.2 192.168.1.3

AN
Auto-Negotiation

API
Application Programming Interface.

B
Data format `short integer`, signed short integer, in the 8-bit range, e.g. 123.

B*
Data format `short integer list`, a variable number of *short integers* with spaces in between, e.g. 1 3 5 7 9

CG
Connection Group

FCS
The frame check sequence (FCS) is a four-octet cyclic redundancy check (CRC) that allows detection of corrupted data within the entire frame as received on the receiver side. According to the standard, the FCS value is computed as a function of the protected MAC frame fields: source and destination address, length/type field, MAC client data and padding (that is, all fields except the FCS).

GUI
Graphical User Interface

H
Data format `hex`, two hexadecimal digits (8-bit long) prefixed by `0x`, e.g. 0xF7. Some parameters consist of multiples of `hex`, for example 0x1234. They are denoted as `hex<n>` or `H..H`, where <n> is the number of `H`. For example, MAC address is of type `hex6` (`HHHHHH`), and IPv6 address is of type `hex16` (`HHHHHHHHHHHHHHHH`).

H*
Data format `hex list`: a variable number of *hexes* specified using a single `0x` prefix,

e.g. 0xF700ABCD2233.

I

Data format `integer`, signed integer, in the 32-bit range, e.g. -1234567.

I*

Data format `integer list`, a variable number of *integers* with spaces in between, e.g. 1000 3000 5000 7000 9000

I2C

I^2C (Inter-Integrated Circuit, eye-squared-C), alternatively known as I2C or IIC, is a synchronous, multi-controller/multi-target (controller/target), packet switched, single-ended, serial communication bus.

Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

L

Data format `long integer`, signed long integer, in the 64-bit range, e.g. 123456789123.

L*

Data format `long integer list`, a variable number of *long integers* with spaces in between, e.g. 123456789 123456789 123456789

Load Profile

A load profile defines a start time and a duration of each of the ramp-up, steady, and ramp-down phases of a connection group.

LT

Link Training

MPLS

Multiprotocol Label Switching (MPLS) is a routing technique in telecommunications networks that directs data from one node to the next based on labels rather than network addresses.

NRZ

NRZ stands for "Non-Return-to-Zero." It is a binary digital encoding scheme used in data transmission, particularly in the context of digital communications. In NRZ encoding, each bit is represented by one of two discrete voltage levels or signal states, typically referred to as "high" and "low."

O

Data format `owner`, a short string up to 32 ASCII characters used to identify an username for resource reservation.

PAM4

PAM4, or Pulse Amplitude Modulation with 4 levels, is a signaling technique used in digital communication systems, particularly in high-speed data transmission. It is a modulation scheme that allows multiple levels of amplitude values for each symbol or signal element, as opposed to binary signaling schemes like NRZ (Non-Return-to-Zero) that use two levels.

PE
Packet Engine

Perl
PEARL, or Process and experiment automation realtime language, is a computer programming language designed for multitasking and real-time programming.

PRBS
Pseudorandom Binary Sequence is a binary sequence that, while generated with a deterministic algorithm, is difficult to predict and exhibits statistical behavior similar to a truly random sequence.

Python
Python is a high-level, interpreted, general-purpose programming language.

S
Data format `string`, 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”.

Tcl
Tcl is a high-level, general-purpose, interpreted, dynamic programming language.

TCP/IP
The Internet protocol suite, commonly known as TCP/IP, is the set of communication protocols used in the Internet and similar computer networks.

Telnet
Telnet is an application protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection.

TG
Traffic Generation

TID
Test Payload Identifier. It is used to identify a sending stream.

TPLD
Test Payload Data. Each Xena test packet contains a special proprietary data area called the Test Payload Data, which contains various information about the packet. The TPLD is located just before the Ethernet FCS.

VLAN
Virtual local area network (VLAN) is any broadcast domain that is partitioned and isolated in a computer network at the data link layer (OSI layer 2).

XLA
Xena Link Analysis

XOA
Xena OpenAutomation.

XOA CLI
: XOA Command-Line Interface. Xena provides a rich set of CLI commands for users to administer test chassis for test automation.

XOA Python API
: The foundation of Xena OpenAutomation is its Python API (XOA Python API) that provides interfaces for engineers to manage Xena hardware and virtual test equipment.

- search

INDEX

M

N

O

P

S

T

V

X