Xena OpenAutomation Python API Documentation

Release 2.5.4

Teledyne LeCroy Xena

May 13, 2024

TABLE OF CONTENT

This document provides a comprehensive guide to using the Xena OpenAutomation (*XOA*) Python API, including installation instructions, usage examples, and API reference documentation. The documentation is organized by API function and includes detailed descriptions of each function, along with usage examples and expected output.

In addition to the API documentation, 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 Python API and other Xena products.

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

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

- Ability to program with Python language.
- 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 CLI* commands, go to Xena OpenAutomation CLI Command Documentation.

CHAPTER

ONE

INTRODUCTION

Xena OpenAutomation (*XOA*) Python API is a standalone Python library that provides a user-friendly and powerful interface for automating network testing tasks using Xena Networks test equipment. Xena test equipment is a high-performance network test device designed for testing and measuring the performance of network equipment and applications.

The XOA Python API is designed to be easy to use and integrate with other automation tools and frameworks. It provides a comprehensive set of methods and classes for interacting with Xena test equipment, including the ability to create and run complex test scenarios, generate and analyze traffic at line rate, and perform detailed analysis of network performance and behavior.

The XOA Python API simplifies the process of automating network testing tasks using Xena test equipment. It provides a simple, yet powerful, interface for interacting with Xena test equipment using the Python programming language. With the XOA Python API, network engineers and testing professionals can easily create and execute test scenarios, generate and analyze traffic, and perform detailed analysis of network performance and behavior, all while leveraging the power and flexibility of the Python programming language.

Additionally, the XOA Python API goes beyond providing object-oriented APIs and functions for executing test scripts. It seamlessly integrates with CLI commands, enabling users to work with them effortlessly.

Overall, the XOA Python API is a valuable tool for anyone looking to automate their network testing tasks using Xena test equipment. With its simple, yet powerful, interface and support for the Python programming language, the XOA Python API provides a flexible and extensible framework for automating network testing tasks and improving the quality of network infrastructure.

1.1 Differences Between XOA Python API and CLI

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 platform, which provides a framework for automating network testing tasks using Xena test equipment. 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.

The XOA Python API and XOA CLI are both tools for automating network testing tasks using Xena test equipment, but they differ in several ways:

- Interface: The XOA Python API provides a Pythonic interface to interact with Xena test equipment, while the XOA CLI provides a command-line interface to interact with Xena test equipment.
- Programming: The XOA Python API is a library that can be used with the Python programming language to create and execute complex test scenarios, generate and analyze traffic, and perform detailed analysis of network performance and behavior. The XOA CLI is a standalone tool that can be used to interact with Xena test equipment through the command line, without the need for programming.
- Functionality: While both tools can be used to create and execute test scenarios, generate and analyze traffic, and perform detailed analysis of network performance and behavior, the XOA Python API provides a more comprehensive and flexible set of functions for interacting with Xena test equipment. The XOA CLI provides a subset of the functionality available through the XOA Python API.
- Ease of use: The XOA Python API provides a more user-friendly and intuitive interface for interacting with Xena test equipment, while the XOA CLI can be more complex and requires knowledge of the command line interface.

1.2 Synergy Between XOA Python API and CLI

Important: Starting from **v2.1.1**, the XOA Python API provides seamlessly integration with CLI commands and port configuration files from ValkyrieManager.

The synergy between the XOA Python API and XOA CLI lies in their integration capabilities. The XOA Python API seamlessly integrates with the XOA CLI, enabling users to work with CLI commands effortlessly within their Python scripts. This integration allows users to combine the flexibility and extensibility of the Python language with the precise control and configuration offered by the CLI commands.

The XOA Python API allows users to interact with Xena Networks test equipment using Python code, providing an object-oriented and user-friendly interface for automating network testing tasks. It enables users to create and execute test scenarios, generate traffic, and analyze network performance using Python programming language. On the other hand, the XOA CLI allows users to configure and control Xena test equipment through command-line commands. It provides a familiar and efficient way to interact with the equipment, allowing users to perform various configuration tasks, manage ports, and execute test commands.

By leveraging both the XOA Python API and XOA CLI, users can take advantage of the best of both worlds. They can harness the power of Python for automation, scripting, and advanced data analysis while utilizing the precise control and configuration options provided by the CLI commands. With the XOA Python API, users can seamlessly work with CLI commands and port configuration files from ValkyrieManager, streamlining the configuration process. Whether users prefer a programming approach or a straightforward command-line interface, both options

are available to suit different requirements and preferences when working with Xena test equipment. This synergy enhances the overall testing experience, enabling users to perform complex testing tasks efficiently and effectively.

In summary, the XOA Python API and XOA CLI work together to provide a comprehensive and flexible testing solution. The Python API brings automation and scripting capabilities, while the CLI offers precise control and configuration options. The integration between the two allows users to leverage their respective strengths and achieve a synergistic testing workflow.

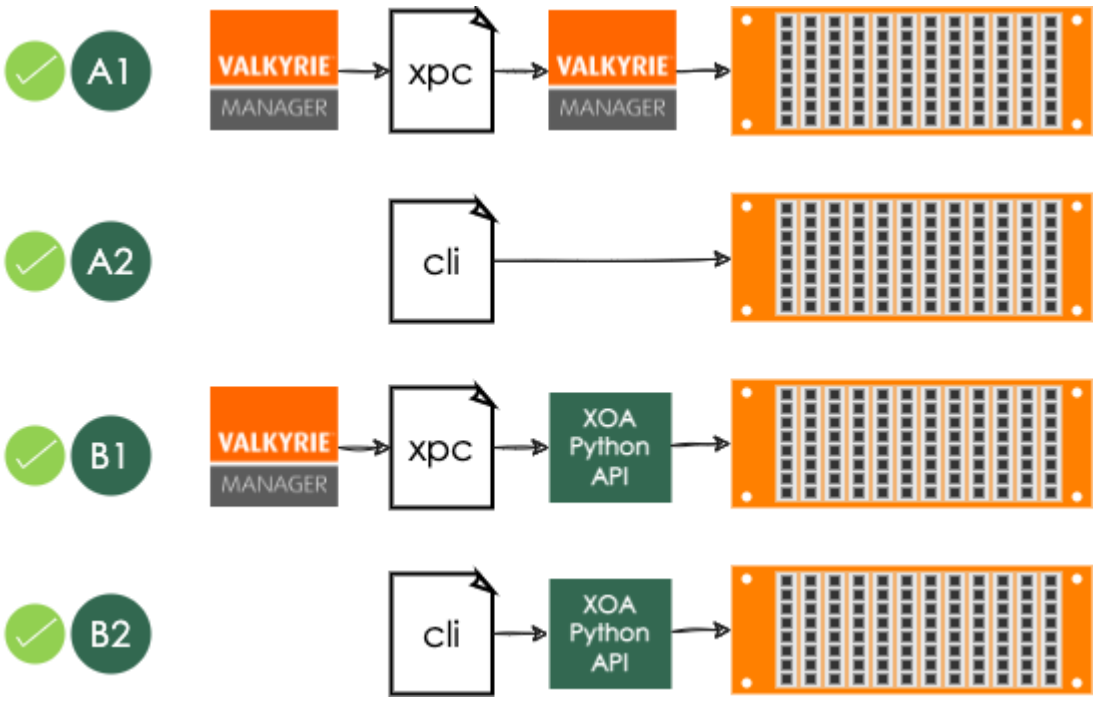


Fig. 1: Synergy Between XOA Python API and CLI

A1: Save port configurations from ValkyrieManager and conveniently load them at a later time.

A2: Use CLI commands to manage and control testers.

B1: Save port configurations from ValkyrieManager and conveniently load them using XOA Python API to facilitate the automation process.

B2: Use CLI commands inside XOA Python API to manage and control testers.

CHAPTER

TWO

GETTING STARTED

2.1 Installing XOA Python API

XOA Python API is available to install and upgrade via the Python Package Index. Alternatively, you can also install and upgrade from the source file.

Note: The minimum Valkyrie release supported by XOA Python API is **83.2**.

2.1.1 Prerequisites

Before installing XOA Python API, please make sure your environment has installed Python and `pip`.

Python

XOA Python API requires that you install Python on your system.

Note: XOA Python API requires Python >= 3.8.

`pip`

Make sure `pip` is installed on your system. `pip` is the package installer for Python . You can use it to install packages from the Python Package Index and other indexes.

Usually, `pip` is automatically installed if you are:

- working in a virtual Python environment (virtualenv or venv). It is not necessary to use `sudo pip` inside a virtual Python environment.
- using Python downloaded from python.org

If you don't have `pip` installed, you can:

- Download the script, from https://bootstrap.pypa.io/get-pip.py.
- Open a terminal/command prompt, `cd` to the folder containing the `get-pip.py` file and run:

Listing 1: Install pip in Windows environment.

```
> py get-pip.py
```

Listing 2: Install pip in macOS/Linux environment.

```
$ python3 get-pip.py
```

See also:

Read more details about this script in pypa/get-pip.

Read more about installation of `pip` in pip installation.

2.1.2 Installing From PyPI Using `pip`

`pip` is the recommended installer for XOA Python API. The most common usage of `pip` is to install from the Python Package Index using Requirement Specifiers.

Note: If you install XOA Core using `pip install xoa-core`, XOA Python API (PyPI package name xoa_driver) will be automatically installed.

Install to Global Namespace

Listing 3: Install XOA Python API in Windows environment from PyPi.

```
> pip install xoa-driver            # latest version
> pip install xoa-driver==1.0.7     # specific version
> pip install xoa-driver>=1.0.7     # minimum version
```

Listing 4: Install XOA Python API in macOS/Linux environment from PyPi.

```
$ pip install xoa-driver            # latest version
$ pip install xoa-driver==1.0.7     # specific version
$ pip install xoa-driver>=1.0.7     # minimum version
```

Install in Activated Virtual Environment

Install XOA Python API in a virtual environment, so it does not pollute your global namespace.

For example, your project folder is called `/my_xoa_project`.

Listing 5: Install XOA Python API in a virtual environment in Windows from PyPI.

```
[my_xoa_project]> python -m venv .\env
[my_xoa_project]> .\env\Scripts\activate

(env) [my_xoa_project]> pip install xoa-driver
```

Listing 6: Install XOA Python API in a virtual environment in macOS/Linux from PyPI.

```
[my_xoa_project]$ python3 -m venv ./env
[my_xoa_project]$ source ./env/bin/activate

(env) [my_xoa_project]$ pip install xoa-driver
```

See also:

- Virtual Python environment
- virtualenv
- venv

Deactivate Virtual Environment

You can deactivate a virtual environment by typing `deactivate` in your shell.

Listing 7: Deactivate virtual environment on Windows.

```
(env) [my_xoa_project]> deactivate
[my_xoa_project]>
```

Listing 8: Deactivate virtual environment on macOS/Linux.

```
(env) [my_xoa_project]$ deactivate
[my_xoa_project]$
```

2.1.3 Upgrading From PyPI Using `pip`

To upgrade XOA Python API package from PyPI:

Listing 9: Upgrade XOA Python API in Windows environment from PyPi.

```
> pip install xoa-driver --upgrade
```

Listing 10: Upgrade XOA Python API in macOS/Linux environment from PyPi.

```
$ pip install xoa-driver --upgrade
```

Note: If you upgrade XOA Core using `pip install --upgrade xoa-core`, XOA Python API (PyPI package name xoa_driver) will be automatically upgraded.

2.1.4 Installing Manually From Source

If for some reason you need to install or upgrade XOA Python API manually from source, the steps are:

Step 1, make sure Python packages wheel and setuptools are installed on your system. Install `wheel` and `setuptools` using `pip`:

Listing 11: Install `wheel` and `setuptools` in Windows environment.

```
> pip install wheel setuptools
```

Listing 12: Install `wheel` and `setuptools` in macOS/Linux environment.

```
$ pip install wheel setuptools
```

Step 2, download the XOA Python API source distribution from XOA Python API Releases. Unzip the archive and run the `setup.py` script to install the package:

Listing 13: Install XOA Python API in Windows environment from source.

```
[xoa_driver]> python setup.py install
```

Listing 14: Install XOA Python API in macOS/Linux environment from source.

```
[xoa_driver]$ python3 setup.py install
```

Step 3, if you want to distribute, you can build `.whl` file for distribution from the source:

Listing 15: Build XOA Python API wheel in Windows environment for distribution.

```
[xoa_driver]> python setup.py bdist_wheel
```

Listing 16: Build XOA Python API wheel in macOS/Linux environment for distribution.

```
[xoa_driver]$ python3 setup.py bdist_wheel
```

Important: If you install XOA Core from the source code, you need to install XOA Python API (PyPI package name xoa_driver) separately. This is because XOA Python API is treated as a 3rd-party dependency of XOA Core. You can go to XOA Python API repository to learn how to install it.

2.1.5 Uninstall and Remove Unused Dependencies

`pip uninstall xoa-driver` can uninstall the package itself but not its dependencies. Leaving the package's dependencies in your environment can later create conflicting dependencies problem.

We recommend install and use the pip-autoremove utility to remove a package plus unused dependencies.

Listing 17: Uninstall XOA Python API in Windows environment.

```
> pip install pip-autoremove
> pip-autoremove xoa-driver -y
```

Listing 18: Uninstall XOA Python API in macOS/Linux environment.

```
$ pip install pip-autoremove
$ pip-autoremove xoa-driver -y
```

See also:

See the pip uninstall reference.

See pip-autoremove usage.

2.2 Quick Start

The XOA Python API offers more than just object-oriented APIs and functions for executing test scripts. It also provides a seamless integration with CLI commands and port configuration files from ValkyrieManager. , enabling you to effortlessly work with them.

Note: Integration with CLI commands and ValkyrieManager is supported by version >= 2.1.1.

2.2.1 Scripting with XOA Python API

The simple code example demonstrates some basics of using *HL-API* and :term: *HL-FUNC*:

- Establish connection to a Valkyrie tester.
- Reserve a port.
- Create a stream on the port.
- Configure the stream.
- Start traffic.
- Collect statistics.
- Stop traffic

We will first walk you through step-by-step covering the topics above. At the end, you will see the whole example. If you want to try it out, you can simply copy and paste it into your environment and run. Remember to change the IP address to your tester's.

This is boilerplate.

```
import asyncio

from xoa_driver import testers
from xoa_driver import modules
```

(continues on next page)

(continued from previous page)

```
from xoa_driver import ports
from xoa_driver import enums
from xoa_driver import utils
from xoa_driver.hlfuncs import mgmt

async def my_awesome_func():
def main():
    try:
        loop = asyncio.get_event_loop()
        loop.create_task(my_awesome_func())
        loop.run_forever()
    except KeyboardInterrupt:
        pass

if __name__ == "__main__":
    main()
```

To establish a connection to a tester is simple.

```
    # Establish connection to a Valkyrie tester 10.10.10.10 with␣
↪username JonDoe.
    async with testers.L23Tester("10.10.10.10", "xoa") as tester:
```

Access module index 0 on the tester. The method `obtain()` is for accessing a test resource that cannot be deleted, such as a module or a port. You can read more about this method in *Module Manager and Port Manager*.

```
        # Access module index 0 on the tester
        my_module = tester.modules.obtain(0)
```

You need to check the type of the test module afterwards, so the driver can allow you to access the methods and attributes of module.

```
        if isinstance(my_module, modules.ModuleChimera):
            return None # commands which used in this example are not␣
↪supported by Chimera Module
```

After that, the driver knows you are using the desired module, and then you can access ports on the module. Let's use two ports, one as TX, the other RX.

```
        # Get the port 0 on module 0 as TX port
        my_tx_port = my_module.ports.obtain(0)
        # Get the port 1 on module 0 as RX port
        my_rx_port = my_module.ports.obtain(1)

        # Reserve the TX port and reset it.
        await mgmt.reserve_port(my_tx_port)
```

(continues on next page)

(continued from previous page)

```
        await mgmt.reset_port(my_tx_port)

        # Reserve the RX port and reset it.
        await mgmt.reserve_port(my_rx_port)
        await mgmt.reset_port(my_rx_port)
```

Now we have two ports ready to configure. Let's start creating a stream on the TX port.

```
        # Create a stream on the TX port
        my_stream = await my_tx_port.streams.create()
        my_tpld_value = 0

        # Prepare stream header protocol
        header_protocol = [enums.ProtocolOption.ETHERNET, enums.
↪ProtocolOption.IP]

        # Simple batch configure the stream on the TX port
        await utils.apply(
            my_stream.tpld_id.set(my_tpld_value), # Create the TPLD␣
↪index of stream
            my_stream.packet.length.set(length_type=enums.LengthType.
↪FIXED, min_val=1000, max_val=1000), # Configure the packet size to␣
↪fixed 1000 bytes
            my_stream.packet.header.protocol.set(header_protocol), #␣
↪Configure the packet type
            my_stream.enable.set_on(), # Enable streams
            my_stream.rate.fraction.set(1000000) # Configure the␣
↪stream rate 100% (1,000,000 ppm)
        )
```

The `await utils.apply()` lets us group several commands bound for the same port into a larger "command". This is called *Sequential Grouping*.

Then, we want to clear the statistics counters of both TX and RX ports. We can use *Parallel Grouping* to group commands bound for different ports into a larger "command".

```
        # Batch clear statistics on TX and RX ports
        await asyncio.gather(
            my_tx_port.statistics.tx.clear.set(),
            my_tx_port.statistics.rx.clear.set(),
            my_rx_port.statistics.tx.clear.set(),
            my_rx_port.statistics.rx.clear.set()
        )
```

Now, let's start the traffic on the TX port for roughly 10 seconds and stop. It is "*roughly*" because we use `sleep()` to control the duration. It may feel accurate to you but for a Valkyrie tester that can generate 800Gbps traffic with time measurement to nanosecond range, `sleep()` is far from accurate in terms of time controlling. If your test requires high-accuracy time control,

don't use software to control time. Instead, limit the port's TX time so that you can have down to microsecond-range traffic duration.

```
        # Start traffic on the TX port
        await my_tx_port.traffic.state.set_start()

        # Test duration 10 seconds
        await asyncio.sleep(10)

        # Stop traffic on the TX port
        await my_tx_port.traffic.state.set_stop()
```

After the traffic is stopped, we query statistic counters. You can also query counter as the traffic is running to get live statistics.

```
        # Wait 2 seconds for the counters to finish
        await asyncio.sleep(2)

        # Query TX statistics
        tx_total, tx_stream = await utils.apply(
            my_tx_port.statistics.tx.total.get(),

            # let the resource manager tell you the stream index so␣
↪you don't have to remember it
            my_tx_port.statistics.tx.obtain_from_stream(my_stream).
↪get()
        )
        print(f"Total TX byte count since cleared: {tx_total.byte_
↪count_since_cleared}")
        print(f"Total TX packet count since cleared: {tx_total.packet_
↪count_since_cleared}")
        print(f"Stream 0 TX byte count since cleared: {tx_stream.byte_
↪count_since_cleared}")
        print(f"Stream 0 TX packet count since cleared: {tx_stream.
↪packet_count_since_cleared}")

        # if you have forgot what TPLD ID assigned to a stream, you␣
↪can query it
        tpld_obj = await my_stream.tpld_id.get()
        # then access the RX stat object
        rx_stats_obj = my_rx_port.statistics.rx.access_tpld(tpld_obj.
↪test_payload_identifier)
        # then query each stats of a TPLD ID
        rx_total, rx_traffic, rx_latency, rx_jitter, rx_error = await␣
↪utils.apply(
            my_rx_port.statistics.rx.total.get(),
            rx_stats_obj.traffic.get(),
            rx_stats_obj.latency.get(),
```

(continues on next page)

(continued from previous page)

```
            rx_stats_obj.jitter.get(),
            rx_stats_obj.errors.get()
        )

        print(f"Total RX byte count since cleared: {rx_total.byte_
↪count_since_cleared}")
        print(f"Total RX packet count since cleared: {rx_total.packet_
↪count_since_cleared}")
        print(f"Stream 0 RX byte count since cleared: {rx_traffic.byte_
↪count_since_cleared}")
        print(f"Stream 0 RX packet count since cleared: {rx_traffic.
↪packet_count_since_cleared}")
        print(f"Stream 0 RX min latency: {rx_latency.min_val}")
        print(f"Stream 0 RX max latency: {rx_latency.max_val}")
        print(f"Stream 0 RX avg latency: {rx_latency.avg_val}")
        print(f"Stream 0 RX min jitter: {rx_jitter.min_val}")
        print(f"Stream 0 RX max jitter: {rx_jitter.max_val}")
        print(f"Stream 0 RX avg jitter: {rx_jitter.avg_val}")
        print(f"Stream 0 RX number of non-incrementing-sequence-number␣
↪events: {rx_error.non_incre_seq_event_count}")
        print(f"Stream 0 RX number of swapped-sequence-number misorder␣
↪events: {rx_error.swapped_seq_misorder_event_count}")
        print(f"Stream 0 RX number of packets with non-incrementing␣
↪payload content: {rx_error.non_incre_payload_packet_count}")
```

At last, release the ports (It is absolutely OK if you don't release them.)

```
        # Release the ports
        await asyncio.gather(
            my_tx_port.reservation.set_release(),
            my_rx_port.reservation.set_release()
        )
```

The entire example is here.

Listing 19: Quick start for some basic.

```
import asyncio

from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports
from xoa_driver import enums
from xoa_driver import utils
from xoa_driver.hlfuncs import mgmt

async def my_awesome_func():
```

(continues on next page)

(continued from previous page)

```
    # Establish connection to a Valkyrie tester 10.10.10.10 with␣
↪username JonDoe.
    async with testers.L23Tester("10.10.10.10", "xoa") as tester:

        # Access module index 0 on the tester
        my_module = tester.modules.obtain(0)

        if isinstance(my_module, modules.ModuleChimera):
            return None # commands which used in this example are not␣
↪supported by Chimera Module

        # Get the port 0 on module 0 as TX port
        my_tx_port = my_module.ports.obtain(0)
        # Get the port 1 on module 0 as RX port
        my_rx_port = my_module.ports.obtain(1)

        # Reserve the TX port and reset it.
        await mgmt.reserve_port(my_tx_port)
        await mgmt.reset_port(my_tx_port)

        # Reserve the RX port and reset it.
        await mgmt.reserve_port(my_rx_port)
        await mgmt.reset_port(my_rx_port)

        # Create a stream on the TX port
        my_stream = await my_tx_port.streams.create()
        my_tpld_value = 0

        # Prepare stream header protocol
        header_protocol = [enums.ProtocolOption.ETHERNET, enums.
↪ProtocolOption.IP]

        # Simple batch configure the stream on the TX port
        await utils.apply(
            my_stream.tpld_id.set(my_tpld_value), # Create the TPLD␣
↪index of stream
            my_stream.packet.length.set(length_type=enums.LengthType.
↪FIXED, min_val=1000, max_val=1000), # Configure the packet size to␣
↪fixed 1000 bytes
            my_stream.packet.header.protocol.set(header_protocol), #␣
↪Configure the packet type
            my_stream.enable.set_on(), # Enable streams
            my_stream.rate.fraction.set(1000000) # Configure the␣
↪stream rate 100% (1,000,000 ppm)
        )
```

(continues on next page)

(continued from previous page)

```
        # Batch clear statistics on TX and RX ports
        await asyncio.gather(
            my_tx_port.statistics.tx.clear.set(),
            my_tx_port.statistics.rx.clear.set(),
            my_rx_port.statistics.tx.clear.set(),
            my_rx_port.statistics.rx.clear.set()
        )

        # Start traffic on the TX port
        await my_tx_port.traffic.state.set_start()

        # Test duration 10 seconds
        await asyncio.sleep(10)

        # Stop traffic on the TX port
        await my_tx_port.traffic.state.set_stop()

        # Wait 2 seconds for the counters to finish
        await asyncio.sleep(2)

        # Query TX statistics
        tx_total, tx_stream = await utils.apply(
            my_tx_port.statistics.tx.total.get(),

            # let the resource manager tell you the stream index so␣
↪you don't have to remember it
            my_tx_port.statistics.tx.obtain_from_stream(my_stream).
↪get()
        )
        print(f"Total TX byte count since cleared: {tx_total.byte_
↪count_since_cleared}")
        print(f"Total TX packet count since cleared: {tx_total.packet_
↪count_since_cleared}")
        print(f"Stream 0 TX byte count since cleared: {tx_stream.byte_
↪count_since_cleared}")
        print(f"Stream 0 TX packet count since cleared: {tx_stream.
↪packet_count_since_cleared}")

        # if you have forgot what TPLD ID assigned to a stream, you␣
↪can query it
        tpld_obj = await my_stream.tpld_id.get()
        # then access the RX stat object
        rx_stats_obj = my_rx_port.statistics.rx.access_tpld(tpld_obj.
↪test_payload_identifier)
        # then query each stats of a TPLD ID
```

(continues on next page)

(continued from previous page)

```
        rx_total, rx_traffic, rx_latency, rx_jitter, rx_error = await␣
↪utils.apply(
            my_rx_port.statistics.rx.total.get(),
            rx_stats_obj.traffic.get(),
            rx_stats_obj.latency.get(),
            rx_stats_obj.jitter.get(),
            rx_stats_obj.errors.get()
        )

        print(f"Total RX byte count since cleared: {rx_total.byte_
↪count_since_cleared}")
        print(f"Total RX packet count since cleared: {rx_total.packet_
↪count_since_cleared}")
        print(f"Stream 0 RX byte count since cleared: {rx_traffic.byte_
↪count_since_cleared}")
        print(f"Stream 0 RX packet count since cleared: {rx_traffic.
↪packet_count_since_cleared}")
        print(f"Stream 0 RX min latency: {rx_latency.min_val}")
        print(f"Stream 0 RX max latency: {rx_latency.max_val}")
        print(f"Stream 0 RX avg latency: {rx_latency.avg_val}")
        print(f"Stream 0 RX min jitter: {rx_jitter.min_val}")
        print(f"Stream 0 RX max jitter: {rx_jitter.max_val}")
        print(f"Stream 0 RX avg jitter: {rx_jitter.avg_val}")
        print(f"Stream 0 RX number of non-incrementing-sequence-number␣
↪events: {rx_error.non_incre_seq_event_count}")
        print(f"Stream 0 RX number of swapped-sequence-number misorder␣
↪events: {rx_error.swapped_seq_misorder_event_count}")
        print(f"Stream 0 RX number of packets with non-incrementing␣
↪payload content: {rx_error.non_incre_payload_packet_count}")

        # Release the ports
        await asyncio.gather(
            my_tx_port.reservation.set_release(),
            my_rx_port.reservation.set_release()
        )

def main():
    try:
        loop = asyncio.get_event_loop()
        loop.create_task(my_awesome_func())
        loop.run_forever()
    except KeyboardInterrupt:
        pass

if __name__ == "__main__":
    main()
```

2.2.2 Integrate with CLI and ValkyrieManager

The simple code example demonstrates how to use XOA Python API :

- Establish connection to a Valkyrie tester.
- Reserve a port.
- Port configuration from *.xpc* file
- Port configuration from CLI commands
- Module configuration from file
- Module configuration from CLI commands
- Chassis configuration from file
- Chassis configuration from CLI commands

We will first walk you through step-by-step covering the topics above. At the end, you will see the whole example. If you want to try it out, you can simply copy and paste it into your environment and run. Remember to change the IP address to your tester's.

This is boilerplate.

```
import asyncio

from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports
from xoa_driver import enums
from xoa_driver import utils
from xoa_driver.hlfuncs import mgmt, cli

async def my_awesome_func(stop_event: asyncio.Event):
async def main():
    stop_event = asyncio.Event()
    try:
        await my_awesome_func(stop_event)
    except KeyboardInterrupt:
        stop_event.set()


if __name__ == "__main__":
    asyncio.run(main())
```

To establish a connection to a tester is simple.

```
    # create tester instance and establish connection
    async with testers.L23Tester("10.10.10.10", "xoa") as tester:
```

Access module index 0 on the tester. The method `obtain()` is for accessing a test resource that cannot be deleted, such as a module or a port. You can read more about this method in *Module Manager and Port Manager*.

```
        # access module 0 on the tester
        module = tester.modules.obtain(0)
```

You need to check the type of the test module afterwards, so the driver can allow you to access the methods and attributes of module.

```
        if isinstance(module, modules.ModuleChimera):
            return None
```

After that, the driver knows you are using the desired module, and then you can access ports on the module.

You can use *Save Port Configuration* in ValkyrieManager to download port configuration files, which contain CLI commands inside. To "upload" the port configuration file generated by ValkyrieManager, simply do:

```
        # access port 0 on the module
        port = module.ports.obtain(0)

        # reserve the port
        await mgmt.reserve_port(port=port)

        # configure port with .xpc file generated by ValkyrieManager
        await cli.port_config_from_file(port=port, path="port_config.
↪xpc")
```

In addition to set port configuration from an *xpc* file, you can also send CLI commands using XOA Python API.

```
        # configure port with CLI commands
        await cli.port_config_from_string(
            port=port,
            long_str="""
            P_RESET
            P_COMMENT \"this is a comment\"
            P_MACADDRESS  0xAAAAAABBBB99
            P_IPADDRESS  1.1.1.1 0.0.0.0 0.0.0.0 0.0.0.0
            """)
```

You can set module or chassis configuration in the same way, either from a file or from command strings.

```
        # reserve the module
        await mgmt.reserve_module(module=module)
```

(continues on next page)

(continued from previous page)

```
        # configure module by file
        await cli.module_config_from_file(module=module, path="module_
↪config.txt")

        # configure module with CLI commands
        await cli.module_config_from_string(
            module=module,
            long_str="""
            M_COMMENT \"this is a comment\"
            M_MEDIA  QSFP_DD_NRZ
            M_CFPCONFIGEXT  2 100000 100000
            """)

        # reserve the tester
        await mgmt.reserve_tester(tester=tester)

        # configure module by file
        await cli.tester_config_from_file(tester=tester, path="tester_
↪config.txt")

        # configure module with CLI commands
        await cli.tester_config_from_string(
            tester=tester,
            long_str="""
            C_COMMENT \"this is a comment\"
            C_NAME \"this is a name\"
            """)
```

The entire example is here.

Listing 20: Configuration By CLI

```
import asyncio

from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports
from xoa_driver import enums
from xoa_driver import utils
from xoa_driver.hlfuncs import mgmt, cli

async def my_awesome_func(stop_event: asyncio.Event):
    # create tester instance and establish connection
    async with testers.L23Tester("10.10.10.10", "xoa") as tester:

        # access module 0 on the tester
        module = tester.modules.obtain(0)
```

(continues on next page)

(continued from previous page)

```
        if isinstance(module, modules.ModuleChimera):
            return None

        # access port 0 on the module
        port = module.ports.obtain(0)

        # reserve the port
        await mgmt.reserve_port(port=port)

        # configure port with .xpc file generated by ValkyrieManager
        await cli.port_config_from_file(port=port, path="port_config.
↪xpc")

        # configure port with CLI commands
        await cli.port_config_from_string(
            port=port,
            long_str="""
            P_RESET
            P_COMMENT \"this is a comment\"
            P_MACADDRESS  0xAAAAAABBBB99
            P_IPADDRESS  1.1.1.1 0.0.0.0 0.0.0.0 0.0.0.0
            """)

        # reserve the module
        await mgmt.reserve_module(module=module)

        # configure module by file
        await cli.module_config_from_file(module=module, path="module_
↪config.txt")

        # configure module with CLI commands
        await cli.module_config_from_string(
            module=module,
            long_str="""
            M_COMMENT \"this is a comment\"
            M_MEDIA  QSFP_DD_NRZ
            M_CFPCONFIGEXT  2 100000 100000
            """)

        # reserve the tester
        await mgmt.reserve_tester(tester=tester)

        # configure module by file
        await cli.tester_config_from_file(tester=tester, path="tester_
↪config.txt")
```

(continues on next page)

(continued from previous page)

```
        # configure module with CLI commands
        await cli.tester_config_from_string(
            tester=tester,
            long_str="""
            C_COMMENT \"this is a comment\"
            C_NAME \"this is a name\"
            """)

async def main():
    stop_event = asyncio.Event()
    try:
        await my_awesome_func(stop_event)
    except KeyboardInterrupt:
        stop_event.set()


if __name__ == "__main__":
    asyncio.run(main())
```

CHAPTER

THREE

UNDERSTANDING XOA PYTHON API

3.1 API Structure

XOA Python API consists of three layers on top of the Xena proprietary binary API, as shown below.

High-level Functions (HL-FUNC) provides high-level abstraction functions

High-Level API (HL-API) provides object-oriented APIs.

Low-Level API (LL-API) provides low-level class.

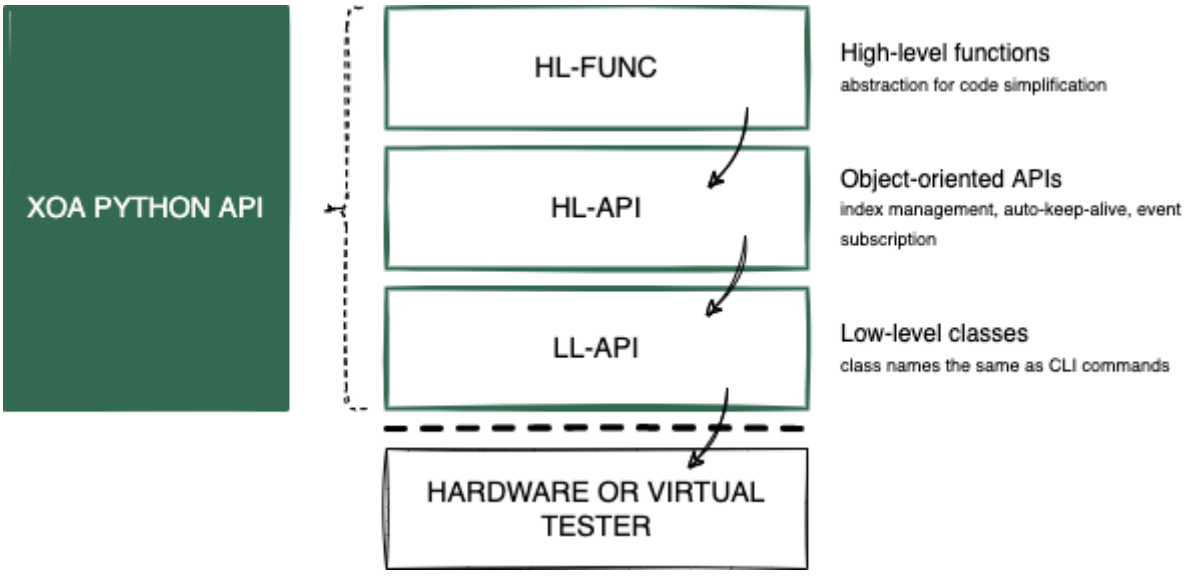


Fig. 1: XOA Python API Stack View

Descriptions of each layer (from bottom to top) are shown below.

Low-Level API

LL-API is the bottom layer containing **low-level command classes** that convert human-readable parameters to and from binary data to communicate testers. The names of the low-level command classes are the same as the the CLI commands in *XOA CLI*. This makes it easy for you to understand and use LL-API if you are already familiar with XOA CLI.

See also:

Read more about *Low-Level API*.

High-Level API

On top of LL-API's command clases, HL-API provides **object-oriented** APIs and lets you quickly develop scripts or programs in an **object-oriented** fashion with explicit definition of commands of different *tester*, *module*, *port* types.

In addition, the HL-API layer provides functionalities such as:

- *Auto connection keep-alive*
- *Auto index management*
- *Resources identification tracking for push notification*

See also:

Read more about *High-Level API*.

High-Level Functions

HL-FUNC provides **high-level abstraction** functions on top of the object-oriented APIs in HL-API, aiming to help you simplify code logics and increase readability and maintainability. HL-FUNC consists of sub-libraries where functions are grouped based on functionalities, such as *ANLT*. Complex operation sequences are wrapped inside high-level functions, e.g. initiating link training, reserving ports, etc.

See also:

Read more about *High-Level Functions*.

3.2 High-Level Functions

HL-FUNC provides high-level abstraction functions on top of the object-oriented APIs in HL-API, aiming to help you simplify code logics and increase readability and maintainability. HL-FUNC consists of sub-libraries where functions are grouped based on functionalities, such as *ANLT*. Complex operation sequences are wrapped inside high-level functions, e.g. initiating link training, reserving ports, etc.

```
from xoa_driver.hlfuncs import anlt, mgmt

# Regardless of who owns the port, this function makes sure you have␣
↪the ownership.
await mgmt.reserve_port(port, force=True)

# Tells the remote link training partner to increase its emphasis␣
↪register value by 1 bit.
await anlt.lt_coeff_inc(port=port, lane=0, emphasis=LinkTrainCoeffs.
↪PRE)
```

The object-oriented APIs in HL-API and the command classes in LL-API are one-to-one mapped, and there is no abstraction provided by the HL-API. As a test specialist, your focus is on building test logics and sequences, not spending unnecessary time on "logistics", such as reserving ports, releasing your ports, deleting all streams on a port without resetting, etc.

To help you simplify code complexity, speed up development, increase readability and maintainability, HL-FUNC is added as the topmost layer, providing abstract functions to the frequently used operations.

3.2.1 Auto-Negotiation and Link Training

Auto-Negotiation and Link Training (ANLT) provides functions to help you fine-tune the protocol to its optimal state, test interoperability between different vendors, and protocol compliance for different implementations.

Auto-negotiation (AN) was originally designed for Ethernet over twisted pair up to 1G. Beyond exchanging speed capabilities for the link participants, AN has evolved for today's Ethernet to include additional configuration information for establishing reliable and consistent connections. AN allows the devices at the end points of a link to negotiate common transmission parameters capabilities like speed and duplex mode, exchange extended page information and media signaling support. At higher speeds and signaling the choice of FEC may be relevant. It is during auto negotiation the end points of a link share their capabilities and choose the highest performance transmission mode they both support.

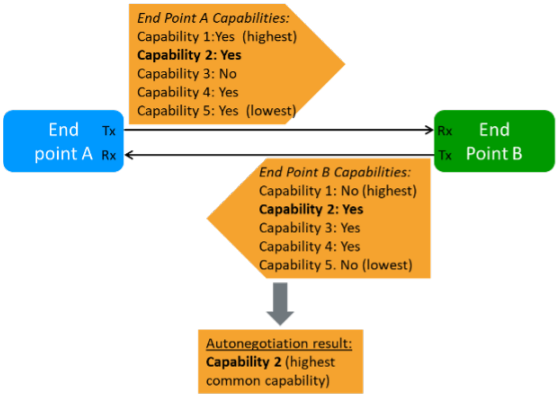


Fig. 2: Auto-Negotiation Process

Once the ports in the link have completed the requisite AN information exchange and reached agreement, the link partners move to the next step, link training (LT), the exchange of Training Sequences. This is essential to tune the channels for optimal transmission. During link training the two end points of the link will exchange signals.

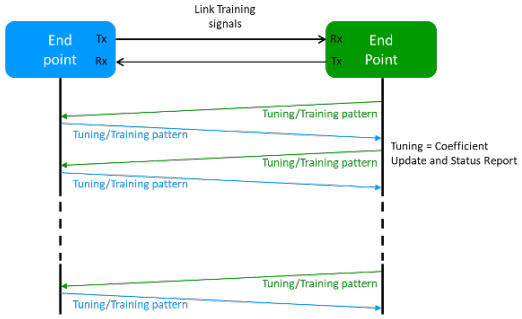


Fig. 3: Link Training Process

No Auto Negotiation, No Link Training

In some instances, Auto negotiation and Link Training are not required to establish a communication path: High speed optical transceivers and interfaces typically only run at one speed, so there is no need the negotiate this. Link Training is only required for electrical interfaces - in some cases (e.g. when short cables are used) an electrical interface may become operational just using default settings of the terminal equipment in the communication path. The IEEE 802.3by specification allows for force connect over electrical interfaces in these instances.

No Auto Negotiation, Link Training

While Link Training can be essential to make some electrical interfaces work, Auto negotiation may not be required is the link speed is fixed or if it can be manually set at both end points of a link.

Auto Negotiation and Link Training

Auto negotiation and Link Training are in principle two independent processes. However, when both are done, Auto negotiation must be done first to determine the overall mode for a link and then perform the Link Training. Hereby you get the sequence shown in the figure below.

Fig. 4: Auto-Negotiation and Link Training Sequence

See also:

Read more about Auto Negotiation and Link Training on NRZ and PAM4 based Ethernet Interfaces.

In HL-FUNC, you can find the following functionalities to do auto-negotiation and link training tests.

AN Functionalities

1. Enable/disable auto-negotiation
2. Auto-negotiation trace log, provides AN trace log for debugging and troubleshooting.
3. Auto-negotiation status, provides the following AN status:
    - Received and transmitted number of Link Code Words (Base Pages), message pages, and unformatted pages
    - Number of HCD (Highest Common Denominator) failures
    - Number of FEC failures
    - Number of LOS (Loss of Sync) failures
    - Number of timeouts
    - Number of successes
    - Duration of AN in microseconds

LT Functionalities

1. Enable/disable link training
2. Allow/deny link training loopback
3. Enable/disable link training timeout
4. Tuning link partner TX EQ coefficient, use presets as a starting point to tune link partner TX EQ coefficients per lane, increment and decrement of coefficients c(-3), c(-2), c(-1), c(0), c(1).
5. Configure local TX EQ coefficients
6. Monitor local TX EQ coefficients
7. Link training trace log per lane
8. Link training status per lane, provides the following LT status:
    - Number of lost locks
    - Local value of coefficient (per coefficient)
    - RX number of increment/decrement requests from link partner (per coefficient)
    - RX number of EQ coefficient request limits reached from link partner (per coefficient)
    - RX number of EQ request limits reached from link partner (per coefficient)
    - RX number of coefficients not supported from link partner (per coefficient)

- RX number of coefficients at limit from link partner (per coefficient)
- TX number of increment/decrement requests to link partner (per coefficient)
- TX number of EQ coefficient request limits reached to link partner (per coefficient)
- TX number of EQ request limits reached to link partner (per coefficient)
- TX number of coefficients not supported to link partner (per coefficient)
- TX number of coefficients at limit to link partner (per coefficient)
- Duration of LT in microseconds
- PRBS total error bits
- PRBS total error bits
- PRBS bit error rate
- Local frame lock status
- Link partner frame lock status

3.2.2 CLI Integration

The XOA Python API allows users to interact with Xena Networks test equipment using Python code, providing an object-oriented and user-friendly interface for automating network testing tasks. It enables users to create and execute test scenarios, generate traffic, and analyze network performance using Python programming language. On the other hand, the XOA CLI allows users to configure and control Xena test equipment through command-line commands. It provides a familiar and efficient way to interact with the equipment, allowing users to perform various configuration tasks, manage ports, and execute test commands.

By leveraging both the XOA Python API and XOA CLI, users can take advantage of the best of both worlds. They can harness the power of Python for automation, scripting, and advanced data analysis while utilizing the precise control and configuration options provided by the CLI commands. With the XOA Python API, users can seamlessly work with CLI commands and port configuration files from ValkyrieManager, streamlining the configuration process. Whether users prefer a programming approach or a straightforward command-line interface, both options are available to suit different requirements and preferences when working with Xena test equipment. This synergy enhances the overall testing experience, enabling users to perform complex testing tasks efficiently and effectively.

1. Send tester configuration from a configuration file
2. Send module configuration from a configuration file
3. Send port configuration from a configuration file (.xpc file)
4. Send tester configuration from a string
5. Send module configuration from a string
6. Send port configuration from a string

3.2.3 Test Resource Management

As described in *Test Resource Management*, you need to reserve the test resource (chassis/module/port) to do *set* operations. In order to achieve this, you need to first check the ownership of the test resource, and relinquish it in case it is owned by someone else, and then reserve it. Such as sequence of operations can be simplified by the high-level abstraction functions in UTIL.

1. Connect to chassis
2. Reserve/Release/Reset ports
3. Reserve/Release chassis (in future release)
4. Reserve/Release module (in future release)
5. Disconnect

3.3 High-Level API

HL-API uses the classes defined in LL-API and lets you quickly develop scripts or program in an **object-oriented** fashion with explicit definition of commands of different *tester*, *module*, *port* types. In addition, the HL-API layer provides functionalities such as:

- *Auto connection keep-alive*
- *Auto index management*
- *Resources identification tracking for push notification*

3.3.1 API Notation and Namings

HL-API aims to be semantic in function naming to avoid expectation conflict, as well as avoiding methods that can return values of different types. The key rule is: **one method, one action**. The following notations are used throughout this chapter.

<resource>
: Represents `Tester | Module | Port | <indices> | <namespace_class>`.

<indices>
: Represents *stream indices*, *connection group indices*, *filter indices*, etc.

<namespace_class>
: A group of commands that manage the resources of the same kind but still stays at the same level as others.

<command_oo_name>
: command name adapted to the object-oriented programming concept. Commands of the same access level, which read or modify parameters of the same type, are grouped under one `<namespace_class>`.

An example of HL-API notation and namings based on the corresponding XOA CLI command names:

Listing 1: CLI command names

```
P_SPEEDSELECTION
P_SPEEDS_SUPPORTED
```

are represented as

Listing 2: Corresponding HL-API naming

```
<resource>.speed.selection
<resource>.speed.supported
```

Note: If there is a method returning both a single value and multiple values, it is considered a bug.

3.3.2 Get and Set Methods

There are only two types of methods for each command, `get` and/or `set`:

- Method `get` is used to **query** the values, status, configuration of the resource.

When you do `get`, you will receive an object that contains the values as its properties. For example, if you want to get the IPv4 configuration of a port, you should do `resp = await port.net_config.ipv4.address.get()` and then variable `resp` will have all the values returned by `P_IPADDRESS`. You can then type `.` after the `resp` and your IDE will show the attributes for you to select as shown below.

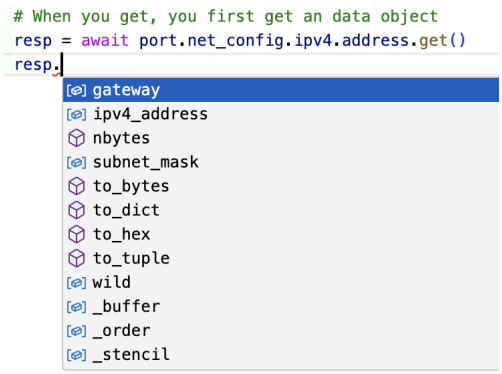


Fig. 5: Auto-complete of method get

- Method `set` is used to **change** the values, status, configuration of the resource.

When you do `set`, your IDE will automatically pop up the expected input arguments and their types as shown below.

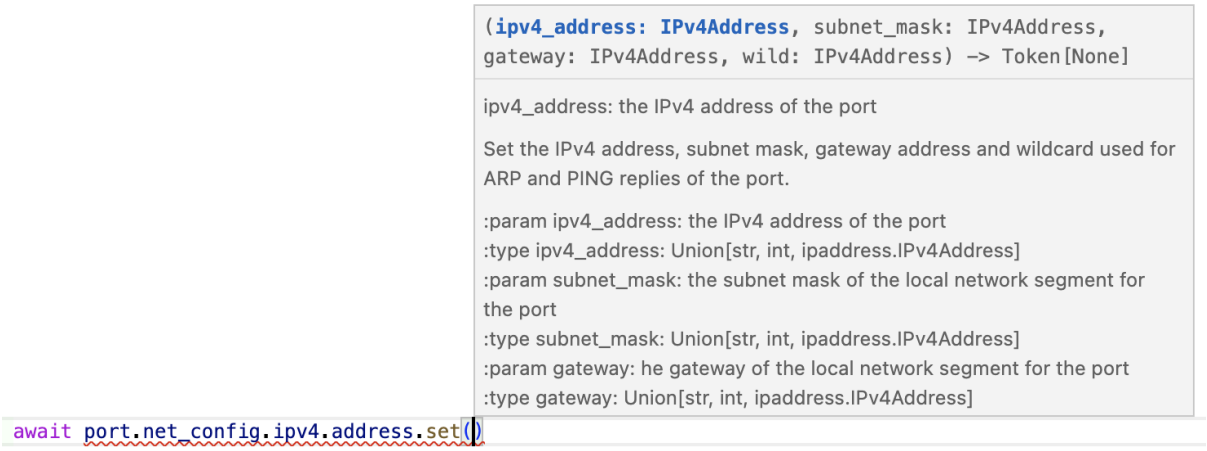


Fig. 6: Auto-complete of method set

Attention: To use `get` and `set` methods, you need to use `await` because they are all made asynchronous.

Syntax:

```
resp = await <resource>.<command_oo_name>.get()

await <resource>.<command_oo_name>.set(<values>)

await <resource>.<command_oo_name>.set_<variation_name>()

await <resource>.<command_oo_name>.set_<variation_name>(<extra_value>)
```

Example:

```
resp = await <Port>.speed.supported.get()

await <Port>.speed.selection.set(mode=PortSpeedMode.AUTO)

await <Port>.<resource>.speed.selection.set_auto()

await <Stream>.packet.length.set_incrementing(min_val=100, max_val=500)
```

See also:

Learn more about Python awaitable object.

3.3.3 Event Subscription and Push Notification

Periodical querying of test resource information, such as port sync statue, is low in communication efficiency and less responsive. Different from XOA CLI, HL-API supports push notification sent from the chassis server when the state or the configuration of a test resource changes. For instance, when a port starts generating traffic, its traffic state is changed from off to on, thus all the connected client programs will receive a push notification message of the new state from the chassis server.

HL-API provides functions for you to subscribe to events, which are triggered test resource state/configuration changes. Thus, your script/application can catch the push notifications and act accordingly.

Syntax:

```
<resource>.on_<command_oo_name>_change(<async_callback_function>)
```

Example:

```
port.on_traffic_change(my_calllback_function)

import asyncio
async def my_calllback_function(port, new_value)
    ...
```

Important: The <async_callback_function> must be a coroutine function

---

Parameters that are passed to your <async_callback_function> depend on the resource it is affiliated:

- Under the Tester level: <ref_tester>, <new_value>
- Under the Module level: <ref_module>, <new_value>
- Under the Port level: <ref_port>, <new_value>

Attention: Exception to the rule above is the event on_disconnected. The parameters passed to it are tuple(<tester_ip: str>, <tester_port: int>)

Note: A subscription to an event only provides a tool for notifying the external code. It is unnecessary to update the library instance state manually, because it is automatically handled by the library code.

It is allowed to subscribe multiple callback functions to one event.

3.3.4 Resource Managers

Most of the subtester resources, which are organized into collections, are handled by *Resource Managers*.

The most commonly used resource managers are *Module Manager and Port Manager* | *Index Managers*.

An illustration of resource managers and *test resources* are shown below:

```
------------------
|     Tester     |
------------------
    |
********************
|  module manager  |
********************
    |
    |   --------------
    |---|  Module 0  |
    |   --------------
    |        |
    |    ********************
    |    |   port manager   |
    |    ********************
    |        |
    |        |    --------------    ******************
    |        |----|  Port 0    | - | index managers |
    |        |    --------------    ******************
    |        |    --------------    ******************
    |        |----|  Port 1    | - | index managers |
    |        |    --------------    ******************
    |        |    --------------    ******************
    |        |----|  Port N-1  | - | index managers |
    |             --------------    ******************
    |
    |   --------------
    |---|  Module 1  |
    |   --------------
    |        |
    |    ********************
    |    |   port manager   |
    |    ********************
    |        |
    |        |    --------------    ******************
    |        |----|  Port 0    | - | index managers |
    |        |    --------------    ******************
    |        |    --------------    ******************
```

(continues on next page)

(continued from previous page)

```
    |         |----|  Port 1     | - | index managers |
    |         |     --------------   ******************
    |         |     --------------   ******************
    |         |----|  Port N-1   | - | index managers |
    |              --------------   ******************
    |
    |   ---------------
    |---| Module N-1 |
        ---------------
```

Note: Each *resource manager* is an iterable object

Module Manager and Port Manager

Each tester object contains a *Module Manager*, which can be accessed through attribute `modules`, e.g. `my_tester.modules`. Each module object contains a *Port Manager*, which can be accessed through attribute `ports`, e.g. `my_module.ports`.

Important: Modules and ports are test resources that cannot be created or deleted, unless the tester is reconfigured either physically or virtually. Thus, in XOA Python API, there is no "create" or "delete" methods for these two types of objects. What we can do is to *obtain* the object that represents the underlying test resource.

A *Module Manager* can contain modules of different *Module Types*. This is because there can be various test modules installed in a physical tester. On the other hand, a *Port Manager* contains ports of the same *Port Type*. This is because the ports on a module are of the same type.

Attention: `obtain()` is not a coroutine function, so don't use `await` with it.

Gain Access to Single Object

Methods to gain access to a module or a port from a *resource manager*:

Syntax:

```
obtain(<module-index> | <port-index>)
```

Gain Access to Multiple Objects

Methods to gain access to multiple resources from a *resource manager*:

Syntax:

```
obtain_multiple(<module-index> | <port-index>, ...)
```

Index Managers

Each port object contains several *Index Managers* that manage the subport-level resource indices such as stream indices, filter indices, connection group indices, modifier indices, etc. It automatically ensures correct and conflict-free **index assignment**.

For L23:

- *Stream Index Manager* can be accessed through attribute `streams`, e.g. `my_l23_port.streams`.
- *Filter Index Manager* can be accessed through attribute `filters`, e.g. `my_l23_port.filters`.
- *Match Term Index Manager* can be accessed through attribute `match_terms`, e.g. `my_l23_port.match_terms`.
- *Length Term Index Manager* can be accessed through attribute `length_terms`, e.g. `my_l23_port.length_terms`.
- *Histogram Dataset Index Manager* can be accessed through attribute `datasets`, e.g. `my_l23_port.datasets`.
- *Modifier Index Manager* can be accessed through attribute `modifiers` under `packet.header` of a stream object, e.g. `my_stream.packet.header.modifiers`

For L47:

- *Connection Group Index Manager* can be accessed through attribute `streams`, e.g. `my_l47_port.connection_groups`.

Important: Streams, connection groups, filters, modifiers, etc. are virtual. They can be created and deleted. Thus in XOA Python API, there are *create*, *delete*, and *remove* methods for you to manage these virtual resources.

It is user's responsibility to create, retrieve, and delete those subport-level indices. Index Managers only takes care of the index assignment.

When you create an index instance under a port, e.g. a stream, the Stream Index Manager will pick an available value and assign it to the stream as the stream index. When you delete an index instance, the index manager will mark that index value as available. When you create an index instance again, the index manager will take the freed values first instead of creating a new one.

This makes sure when the index manager cannot create more index instances is only because of the port capability, not because of the wasted index values.

Thanks to the index assignment mechanism, you don't necessarily need to handle the index assignment but concentrating on the test logic. Methods to manage subport-level instances:

- To create an index, use the method `<index_manager>.create()` under the index manager, e.g. `my_stream = await my_port.streams.create()`.
- To delete an index, you can use the method `<index_manager>.remove(<index>)` under the index manager, e.g. `await my_port.streams.remove(0)`. However, the method `remove` expects the index value of the instance.
- An easier way to delete an index is using method `<index_instance>.delete()` directly on the index instance, e.g. `await my_stream.delete()`. The call of the function `<index_instance>.delete()` will delete the index from the port, and will automatically notify the index manager about the deletion.

3.3.5 Session

A `session` will be created automatically after a TCP connection is established between the client and the tester.

Three attributes of a `session` are exposed:

- `is_online` - property to validate if the TCP connection is alive.
- `logoff()` - async method for gracefully closing the TCP connection to the tester.
- `sessions_info()` - async method for getting information of the current active sessions on a tester.

Session Identification

- A tester does not use the tuple (source IP, source port, destination IP, destination port) to identify a session. Instead, it uses the username as the identification of a session. For instance, `tester = await testers.L23Tester("192.168.1.200", "JonDoe")`, where the username is `JonDoe`.

Session Recovery and Resource Reallocation

- To recover the session, the client only needs to establish a new TCP connection with the same username as the dropped session.
- All resources of the broken session will be automatically transferred to the new session because they have the same username.

Handling Multiple Same-Username Sessions

- If multiple sessions use the same username to connect to a tester after a broken session, the tester will give the control of the resources to a session in a first-come-first-served manner, and the others will be treated as observers. Thus, duplicated username should be avoided at the session level.
- If the controlling session is disconnected, the tester will automatically pass the control of the resources to the next session in the queue.

3.3.6 Local State

The access to the *local state* of a resource is done through property `<resource>.info`. The info contains current status of the resource and information of its attributes, which cannot be changed during a running `session`.

3.4 Low-Level API

LL-API is the bottom layer containing low-level command classes that convert human-readable parameters to and from binary data to communicate testers. The names of the low-level command classes are the same as the the CLI commands in *XOA CLI*. This makes it easy for you to understand and use LL-API if you are already familiar with XOA CLI.

You can use LL-API directly in your test scripts. Using LL-API is similar to CLI scripting where no object-oriented programming mindset is required. Thus, it is easy to start with if you prefer writing test scripts in a CLI fashion, for example:

```
# Directly using class P_RESERVATION. This is only valid when the port␣
↪is not reserved by others.
await P_RESERVATION(handler).set(operation=ReservedAction.RESERVE)
```

However, the trade-off using LL-API directly is that you need to handle the connection keep-alive in your code (no *auto connection keep-alive* feature) and you need to handle the creation and deletion of stream indices, filter indices, modifier indices, etc. (no *auto index management* feature). This means there will be more lines of code in your test scripts.

3.4.1 API Notation and Namings

LL-API aims to be semantic in function naming to avoid expectation conflict, as well as avoiding methods that can return values of different types. The key rule is: **one method, one action**. The following notations are used throughout this chapter.

<indices>
: Represents *stream indices*, *connection group indices*, *filter indices*, etc.

<prefix_command_group>
: A group of commands that manage the resources of the same kind but still

stays at the same level as others. For example, P_SPEEDSELECTION and P_SPEEDS_SUPPORTED are in the P_ category.

<command_name>
The CLI name of the command. Commands of the same access level, which access or modify parameters of the same kind, are grouped under one command group as shown in the example below.

```
P_SPEEDSELECTION
P_SPEEDS_SUPPORTED
```

are represented as

```
P_SPEEDSELECTION(TransportationHandler, <indices>)
P_SPEEDS_SUPPORTED(TransportationHandler, <indices>)
```

Note: If there is a method returning both a single value and multiple values, it is considered a bug.

3.4.2 Attributes and Methods

There are only two types of methods for each command, `get` and/or `set`. `get` is used to query values, status, configuration of the command. `set` is the change.

To use `get` and `set` methods, you need to use `await` because they are all made asynchronous.

Syntax:

```
await <command_name>(TransportationHandler, <indices>).get()

await <command_name>(TransportationHandler, <indices>).set(<values>)
```

Example:

```
await P_SPEEDS_SUPPORTED(TransportationHandler, 0).get()

await P_SPEEDSELECTION(TransportationHandler, 0).set(PortSpeedMode.
↪AUTO)
```

See also:

Read more about Python awaitable object.

CHAPTER

FOUR

TEST RESOURCE MANAGEMENT

If you are new to Xena testers, this section will help you understand the basics of test resource management. *Test resources* can be the chassis itself, a test module on the chassis or a test port on a module.

This section describes:

- *Test resource* hierarchy.
- *Test resource* management principle.

4.1 Test Resource Hierarchy

4.1.1 Valkyrie Tester (L23 Tester)

Valkyrie is a full-featured stateless Ethernet traffic generator and analysis platform. Valkyrie tester has the following hierarchical structure.

```
---------------------
|  Valkyrie Tester  |
---------------------
    |
    |   ----------------------
    |---|   Valkyrie Module   |
    |   ----------------------
    |         |
    |         |    -------------------
    |         |----|  Valkyrie Port  |
    |         |    -------------------
    |         |         |
    |         |         |    ************************
    |         |         |----|  Port Statistics      |
    |         |         |    ************************
    |         |         |    ************************
    |         |         |----|  Stream               |
    |         |         |    ************************
```

(continues on next page)

(continued from previous page)

```
    |         |         |         |
    |         |         |         |    **********************
    |         |         |         |----|  Filter             |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Modifier           |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Histogram          |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Length Term        |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Match Term         |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Test Payload       |
    |         |         |         |    **********************
    |         |         |         |    **********************
    |         |         |         |----|  Stream Statistics  |
    |         |         |         |    **********************
    |         |         |         |
```

Valkyrie Tester, Valkyrie Module, and Valkyrie Port are hardware resources that correspond to the hardware configuration. They cannot be created or deleted.

Everything below Valkyrie Port is virtual resources that can be created, deleted, and configured as needed.

4.1.2 Vulcan Tester (L47 Tester)

Vulcan generates stateful traffic over Ethernet. Vulcan Tester has the following hierarchical structure.

```
------------------
|  Vulcan Tester |
------------------
    |
    |   -------------------
    |---|  Vulcan Module  |
    |   -------------------
    |         |
    |         |    ------------------
    |         |----|  Vulcan Port    |
    |         |    ------------------
```

(continues on next page)

(continued from previous page)

```
    |         |         |
    |         |         |    ************************
    |         |         |----|  Port Statistics      |
    |         |         |    ************************
    |         |         |    ************************
    |         |         |----|  Connection Group     |
    |         |         |    ************************
    |         |         |
```

Vulcan Tester, Vulcan Module, and Vulcan Port are physical resources that correspond to the physical configuration. They cannot be created or deleted.

Everything below Vulcan Port is virtual resources that can be created, deleted, and configured as needed.

4.1.3 VulcanVE Tester (L47VE Tester)

VulcanVE is the virtual edition of Vulcan. VulcanVE Tester has the following hierarchical structure, the same as Vulcan Tester.

```
----------------------
|   VulcanVE Tester  |
----------------------
    |
    |   ----------------------
    |---|   VulcanVE Module  |
    |   ----------------------
    |         |
    |         |    --------------------
    |         |----|   VulcanVE Port  |
    |         |    --------------------
    |         |         |
    |         |         |    ************************
    |         |         |----|  Port Statistics      |
    |         |         |    ************************
    |         |         |    ************************
    |         |         |----|  Connection Group     |
    |         |         |    ************************
    |         |         |
```

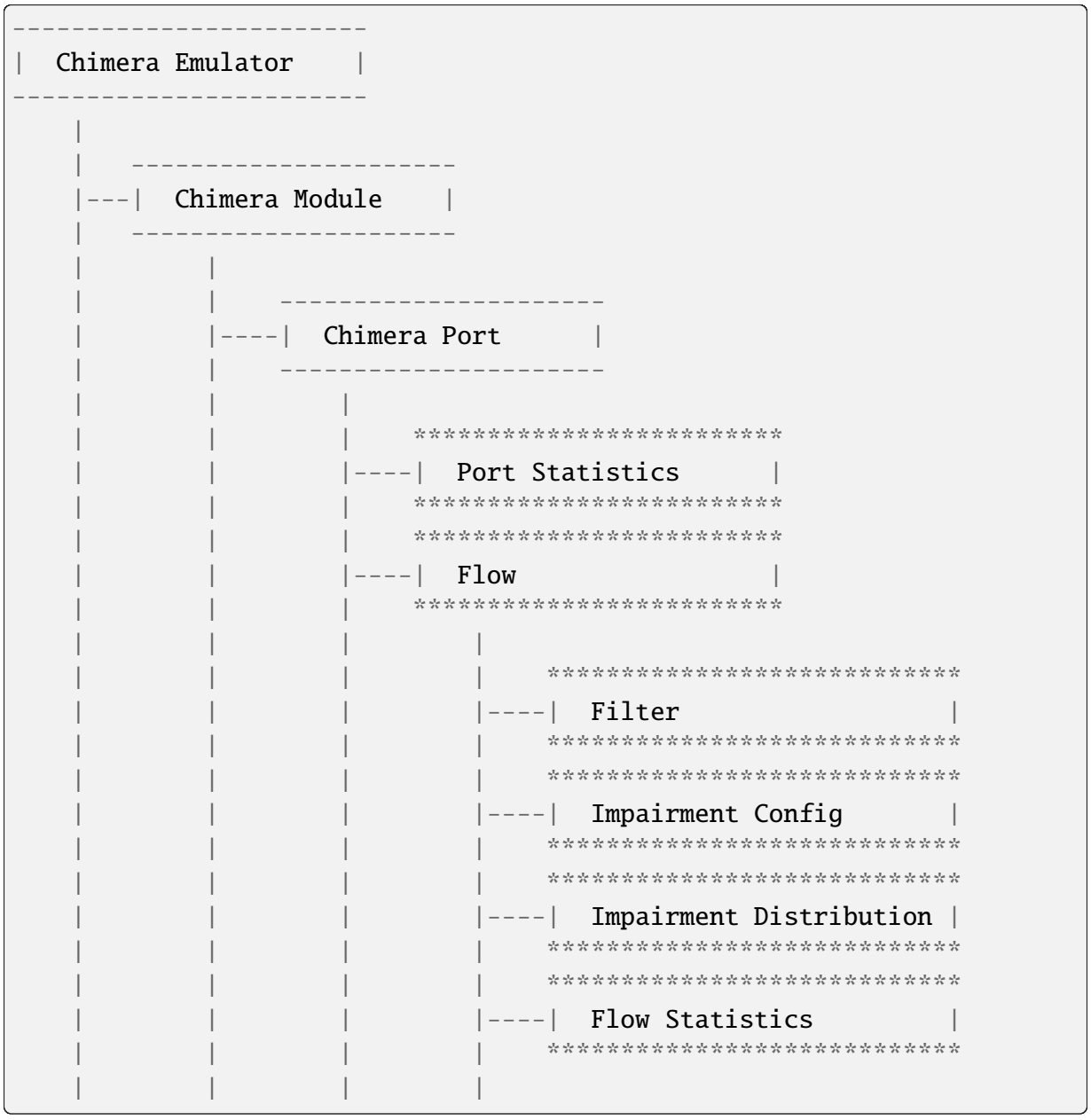
Although VulcanVE Tester, VulcanVE Module, and VulcanVE Port are virtual resources, they cannot be created or deleted.

Everything below VulcanVE Port is virtual resources that can be created, deleted, and configured as needed.

4.1.4 Chimera Network Impairment Emulator (Impairment)

Chimera is a network impairment emulator that makes it easy to introduce consistent, accurate, well-defined and repeatable impairments (e.g. packet manipulation, packet drop, latency and jitter) to traffic between *DUT* in the lab.

Chimera Emulator has the following hierarchical structure.



Chimera Emulator, Chimera Module, and Chimera Port are physical resources that correspond to the physical configuration. They cannot be created or deleted.

Everything below Chimera Port is virtual resources that can be created, deleted, and configured as needed.

Important: Chimera can be seamlessly integrated with Valkyrie by installing Chimera modules in a Valkyrie chassis.

```
---------------------
|  Valkyrie Tester  |
---------------------
    |
    |   ------------------------
    |---|   Valkyrie Module    |
    |   ------------------------
    |
    |   ----------------------
    |---|  Chimera Module    |
    |   ----------------------
```

4.2 Management Principle

Xena testers support multiple simultaneous connections from any mixture of Xena clients, such as the ValkyrieManager, scripting clients, etc. As soon as a client has successfully established a connection to the chassis, any *test resource* can be inspected. But in order to change the *test resource* configuration, the resource must first be reserved by the client.

To management *test resources*, i.e., read, write, create, delete, you must follow the principles below:

1. To do `set` (create/update/delete) on a *test resource*, i.e. *tester*, *module*, or *port*, you must reserve the resource under your username.
2. To do `get` (read) on a *test resource*, you don't need to reserve.
3. To reserve a tester, you must make sure **all the modules and ports are either released or under your ownership**.
4. To reserve a module, you must make sure **all the ports are either released or under your ownership**.

Important: Starting traffic using `C_TRAFFIC` of `C_TRAFFICSYNC` does **NOT** require chassis reservation but port reservation, although their command prefix is `C_` and categorized as chassis-level commands.

CHAPTER

FIVE

COMMAND GROUPING

Using *XOA CLI* to configure ports and streams is slow because a CLI script must wait for a chassis response to before sending the next command. Such a one-by-one fashion results in *N round trip time* (*N-RTT*), where *N* is the number of commands to send.

Because of the abovementioned N-RTT problem, it is difficult for a CLI script to collect traffic statistics of different ports at the same time (using for loops in the script is far from solving the problem). As a result, this will cause a wrong understanding of the test results.

When you are using HL-API or LL-API to develop your test scripts, you can use *Command Grouping* feature to group several commands and send to the tester in one batch.

Depending on the destination the commands are bound for, either to the same or different ports, XOA Python API provides two ways of grouping commands, *Sequential Grouping* (all commands bound for the same port/module) and *Parallel Grouping* (commands are bound for different ports/modules).

5.1 Sequential Grouping

`utils.apply` groups commands in a sequential way. Commands are sent out in one large batch to the tester. This is very useful when you want to send many commands to the same *test resource*, e.g. a port on a tester.

```
commands = [
    command_1,
    command_2,
    command_3,
    ...
]
async for response in utils.apply(*commands):
    print(response)
```

However, abusing this function can cause memory issue on your computer. This is because the computer needs to store all the grouped commands in the memory until the responses from the testers arrive. To avoid potential grouping abuse, a limit of **200** is place to the maximum number of commands that you can group sequentially.

`utils.apply_iter` does exactly the same thing as `utils.apply` except it does not aggregate responses but return them one by one as soon as they are ready. This allows sending large batches commands without causing memory issue.

```
commands = [
    command_1,
    command_2,
    command_3,
    ...
]
async for response in utils.apply_iter(*commands):
    print(response)
```

5.2 Parallel Grouping

`asyncio.gather` groups commands in a parallel way. Commands are sent out in parallel (with neglectable delay between each other). This is very useful when you want to send commands to different *test resources*, e.g. two different ports on the same tester, or two different ports on different testers.

```
await asyncio.gather(
    command_1,
    command_2,
    command_3,
    ...
)
```

5.3 One-By-One

If you prefer sending commands one by one in the same way as using CLI, you can simply place only one command in the group, for example:

```
await command_1
await command_2
await command_3
```

Note: Remember to use `await` before the command. Commands are defined as Coroutines and must be awaited.

See also:

Read more about Python awaitable object.

CHAPTER

SIX

STATUS MESSAGES AND EXCEPTIONS

When you do a `set` operation, XOA Python API converts it into a request message (binary encoded) and send to the server on the tester. Upon receiving the message, the server tries to execute it and returns a **status message** for you to check whether the `set` operation is successful. The returned message may cause an **exception** if it is not an <OK>.

6.1 Status Messages

The `set` operations themselves simply produce a reply from the tester of: <OK>

In case something is unacceptable to the tester, it will return one of the following status messages. In XOA Python API, all of them are considered as `BadStatus`.

- <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.

6.2 Exceptions

If the status message from the server is not <OK>, an exception will be raised by XOA Python API. An example of an exception caused by a <NOTWRITABLE> reply is shown here:

```
Traceback (most recent call last):
File "example.py", line 128, in <module>
    asyncio.run(main())
File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.
↪10/asyncio/runners.py", line 44, in run
    return loop.run_until_complete(main)
File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.
↪10/asyncio/base_events.py", line 641, in run_until_complete
    return future.result()
File "example.py", line 122, in main
    await my_awesome_func(stop_event)
File "example.py", line 89, in my_awesome_func
    await my_port.eee.mode.set_off()
File "/env/lib/python3.10/site-packages/xoa_driver/internals/core/
↪transporter/token.py", line 36, in __ask
    raise e
File "/env/lib/python3.10/site-packages/xoa_driver/internals/core/
↪transporter/token.py", line 34, in __ask
    result = await fut
xoa_driver.internals.core.transporter.exceptions.BadStatus: Bad status
↪<CommandStatus.NOTWRITABLE: 4> of P_LPTXMODE!

Response              : ['58', '45', '4e', '41', '00', '00', '00', '00
↪', '04', '04', 'ff', 'ff', '00', '00', '01', 'ea']
class_name            : P_LPTXMODE
magic_word            : b'XENA'
number_of_indices     : 0
number_of_value_bytes: 0
command_parameter     : 1028:Replied
module_index          : 255
```

(continues on next page)

(continued from previous page)

```
port_index            : 255
request_identifier    : 490
index_values          : []
values                : None
```

If your script doesn't catch the exception, its execution will be interrupted. Since you won't know what exception may happen before running your script, we **highly recommend you handle exceptions in your script**, especially when you want your script to keep running regardless of the replied status messages received from the server.

6.3 Handling Exceptions

6.3.1 Basic Exception Handling

If you know Python, you can simply write codes to handle exceptions caused by the reply from the server.

```
import asyncio
from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports

async def my_awesome_script():
    tester = await testers.L23Tester(host="10.20.1.253", username="XOA
↪", debug=True)

    my_module = tester.modules.obtain(0)

    if isinstance(my_module, modules.ModuleChimera):
        return None # commands which used in this example are not␣
↪supported by Chimera Module

    if my_module.is_reserved_by_me():
        await my_module.reservation.set_release()
    if not my_module.is_released():
        await my_module.reservation.set_relinquish()
    await my_module.reservation.set_reserve()

    my_port = my_module.ports.obtain(0)

    try:
        await my_port.eee.enable.set_off()
        await my_port.eee.mode.set_off()
    except Exception as e:
        print(e) # You decide how to handle the exception
```

See also:

Read more about Handling Exceptions in Python.

6.3.2 Ignore Exceptions

You can also use context manager `suppress` to **ignore exceptions** if you don't care about the `BadStatus` but just want to run the script.

Note: A very common use case of ignoring exception is when you run your script to configure a port. Some ports may not support all the API calls in your script, and may return <NOTVALID> or <NOTSUPPORTED>. But since your objective is to configure the port whatever it supports, you can ignore the exceptions and keep your script running to the end of it.

```
import asyncio
from contextlib import suppress
from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports
from xoa_driver import exceptions

async def my_awesome_script():
    tester = await testers.L23Tester(host="10.20.1.253", username="XOA
↪", debug=True)

    my_module = tester.modules.obtain(0)

    if isinstance(my_module, modules.ModuleChimera):
        return None # commands which used in this example are not␣
↪supported by Chimera Module

    if my_module.is_reserved_by_me():
        await my_module.reservation.set_release()
    if not my_module.is_released():
        await my_module.reservation.set_relinquish()
    await my_module.reservation.set_reserve()

    my_port = my_module.ports.obtain(0)

    with suppress(exceptions.BadStatus):
        await my_port.eee.enable.set_off()
        await my_port.eee.mode.set_off()

    print(f"your script will ignore the exception BadStatus and␣
↪continue")
```

6.3.3 Show Exceptions In Command Grouping

If you want to do **command grouping** (send multiple commands in one batch) **but at the same time want to know which one(s) raises exception**, you use `asyncio.gather` with `return_exceptions=True` as shown here:

```
import asyncio
from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports

async def my_awesome_script():
    tester = await testers.L23Tester(host="10.20.1.253", username="XOA
↪", debug=True)

    my_module = tester.modules.obtain(0)

    if isinstance(my_module, modules.ModuleChimera):
        return None # commands which used in this example are not␣
↪supported by Chimera Module

    if my_module.is_reserved_by_me():
        await my_module.reservation.set_release()
    if not my_module.is_released():
        await my_module.reservation.set_relinquish()
    await my_module.reservation.set_reserve()

    my_port = my_module.ports.obtain(0)

    responses = asyncio.gather(
        my_port.eee.enable.set_off(),
        my_port.eee.mode.set_off(),
        my_port.capabilities.get(),
        return_exceptions=True
    )
    print(responses)
```

CHAPTER

SEVEN

CODE EXAMPLES

You can find various code examples in XOA Scripting Library.

We recommend you start from Quick Start. Some highlighted examples that you may find helpful:

1. XenaAsyncWrapper: The APIs provided by xoa-driver are **async** functions. This means any function that uses the xoa-driver must be declared as **async**. This might be a problem for you if your existing framework doesn't support async functions. To solve this "incompatibility" issue, we have made an async wrapper class **XenaAsyncWrapper** for you to wrap xoa-driver's async function inside and use it as a regular Python function.
2. Use XOA Python API to load port configuration file as you do on ValkyrieManager.
3. Send CLI via XOA Python API demonstrates how to load `.xpc` file or send CLI commands via XOA Python API.
4. Low-Level API demonstrates how to use xoa-driver's low-level API if you are familiar with XOA CLI.
5. Network Emulation demonstrates how to automate Chimera for network emulation.
6. PPM Sweep and ANLT on Thor demonstrates how to change media configuration, perform PPM sweep and AN&LT on Thor modules.
7. IP ARP/NDP Table demonstrates how to generate ARP/NDP table on a port for IP streams.
8. Packet Capture demonstrates how to capture packets and read their content.

More code examples can be found in XOA Scripting Library.

CHAPTER

EIGHT

API REFERENCE

8.1 High-Level Functions

HL-FUNC provides high-level abstraction functions on top of the object-oriented HL-API, aiming to help you simplify code logics and increase readability and maintainability.

HL-FUNC consists of sub-libraries where functions are grouped based on functionalities, such as *ANLT*. Complex operation sequences are wrapped inside high-level functions, e.g. initiating link training, reserving ports, etc.

8.1.1 Resource Management

The following high-level functions handle test resource management, e.g. connection, port reservation, and port reset.

HL Port Functions

HL Module Functions

HL Tester Functions

8.1.2 Auto-Negotiation and Link Training

The following high-level functions are for auto-negotiation and link training.

HL ANLT Functions

HL Auto-Negotiation Functions

HL Link Training Functions

8.1.3 CLI Integration

You can utilize the following high-level functions to set up testers, modules, and ports by employing CLI commands from command strings or files.

Configure from String

Configure from File

8.2 High-Level API

HL-API uses the classes defined in LL-API and lets you quickly develop scripts or program in an **object-oriented** fashion with explicit definition of commands of different *tester*, *module*, *port* types. In addition, the HL-API layer provides functionalities such as:

- *Auto connection keep-alive*
- *Auto index management*
- *Resources identification tracking for push notification*

Important: To continuously improve the usability of XOA Python API, the HL-API will be restructured, especially for the Layer-1 configuration APIs, in the next major release.

For backward-compatibility, the current HL-API is marked as V1. The restructured will be called V2.

You don't need to do change to your import path or code if you continue to use HL-API V1. Both versions will keep being maintained and supported.

The restructuring won't affect the LL-API.

The HL-API (V1) are categorized into:

8.2.1 Summary

The summary of XOA HL-API (object-oriented) is provided in the attached file.

You can find `set` and `get` method of APIs of `tester`, `module`, `port`, `stream`, and `impairment flow`.

See also:

Read more about *Get and Set Method*

```
import asyncio

from xoa_driver import testers
from xoa_driver import modules
from xoa_driver import ports
from xoa_driver import enums
from xoa_driver import utils
from xoa_driver import misc
from xoa_driver.hlfuncs import mgmt
from xoa_driver.misc import Hex
from xoa_driver.lli import commands
import ipaddress


async def my_awesome_func(stop_event: asyncio.Event):

# region Tester
    ##################################################
    #                    Tester                      #
    ##################################################
    tester = await testers.L23Tester(
        host="10.10.10.10",
        username="my_name",
        password="xena",
        enable_logging=False)

    # Shutdown/Restart
    await tester.down.set(operation=enums.ChassisShutdownAction.POWER_
↪OFF)
    await tester.down.set_poweroff()
    await tester.down.set(operation=enums.ChassisShutdownAction.
↪RESTART)
    await tester.down.set_restart()

    # Flash
    await tester.flash.set(on_off=enums.OnOff.OFF)
    await tester.flash.set_off()
    await tester.flash.set(on_off=enums.OnOff.ON)
```

(continues on next page)

(continued from previous page)

```
    await tester.flash.set_on()

    resp = await tester.flash.get()
    resp.on_off

    # Debug Log
    resp = await tester.debug_log.get()
    resp.data
    resp.message_length

    # IP Address
    await tester.management_interface.ip_address.set(
        ipv4_address=ipaddress.IPv4Address("10.10.10.10"),
        subnet_mask=ipaddress.IPv4Address("255.255.255.0"),
        gateway=ipaddress.IPv4Address("10.10.10.1"))

    resp = await tester.management_interface.ip_address.get()
    resp.ipv4_address
    resp.subnet_mask
    resp.gateway

    # MAC Address
    resp = await tester.management_interface.macaddress.get()
    resp.mac_address

    # Hostname
    await tester.management_interface.hostname.set(hostname="name")

    resp = await tester.management_interface.hostname.get()
    resp.hostname

    # DHCP
    await tester.management_interface.dhcp.set(on_off=enums.OnOff.ON)
    await tester.management_interface.dhcp.set_on()
    await tester.management_interface.dhcp.set(on_off=enums.OnOff.OFF)
    await tester.management_interface.dhcp.set_off()

    resp = await tester.management_interface.dhcp.get()
    resp.on_off

    # Capabilities
    resp = await tester.capabilities.get()
    resp.version
    resp.max_name_len
    resp.max_comment_len
    resp.max_password_len
```

(continues on next page)

(continued from previous page)

```
    resp.max_ext_rate
    resp.max_session_count
    resp.max_chain_depth
    resp.max_module_count
    resp.max_protocol_count
    resp.can_stream_based_arp
    resp.can_sync_traffic_start
    resp.can_read_log_files
    resp.can_par_module_upgrade
    resp.can_upgrade_timekeeper
    resp.can_custom_defaults
    resp.max_owner_name_length
    resp.can_read_temperatures
    resp.can_latency_f2f

    # Name
    await tester.name.set(chassis_name="name")

    resp = await tester.name.get()
    resp.chassis_name

    # Password
    await tester.password.set(password="xena")

    resp = await tester.password.get()
    resp.password

    # Description
    await tester.comment.set(comment="description")

    resp = await tester.comment.get()
    resp.comment

    # Model
    resp = await tester.model.get()
    resp.model

    # Serial Number
    resp = await tester.serial_no.get()
    resp.serial_number

    # Firmware Version
    resp = await tester.version_no.get()
    resp.chassis_major_version
    resp.pci_driver_version

```

(continues on next page)

(continued from previous page)

```
    resp = await tester.version_no_minor.get()
    resp.chassis_minor_version
    resp.reserved_1
    resp.reserved_2

    # Build String
    resp = await tester.build_string.get()
    resp.build_string

    # Reservation
    await tester.reservation.set(operation=enums.ReservedAction.
↪RELEASE)
    await tester.reservation.set_release()
    await tester.reservation.set(operation=enums.ReservedAction.
↪RELINQUISH)
    await tester.reservation.set_relinquish()
    await tester.reservation.set(operation=enums.ReservedAction.
↪RESERVE)
    await tester.reservation.set_reserve()

    resp = await tester.reservation.get()
    resp.operation

    # Reserved By
    resp = await tester.reserved_by.get()
    resp.username

    # Information
    # The following are pre-fetched in cache when connection is␣
↪established, thus no need to use await

    tester.session.owner_name
    tester.session.keepalive
    tester.session.pwd
    tester.session.is_online
    tester.session.sessions_info
    tester.session.timeout
    tester.is_released()
    tester.is_reserved_by_me()

    # Logoff
    await tester.session.logoff()

    # Time
    resp = await tester.time.get()
    resp.local_time
```

(continues on next page)

(continued from previous page)

```
171
172     # TimeKeeper Configuration
173     await tester.time_keeper.config_file.set(config_file="filename")
174
175     resp = await tester.time_keeper.config_file.get()
176     resp.config_file
177
178     # TimeKeeper GPS State
179     resp = await tester.time_keeper.gps_state.get()
180     resp.status
181
182     # TimeKeeper License File
183     await tester.time_keeper.license_file.set(license_content="")
184
185     resp = await tester.time_keeper.license_file.get()
186     resp.license_content
187
188     # TimeKeeper License State
189     resp = await tester.time_keeper.license_state.get()
190     resp.license_errors
191     resp.license_file_state
192     resp.license_type
193
194     # TimeKeeper Status
195     resp = await tester.time_keeper.status.get()
196     resp.status_string
197
198     resp = await tester.time_keeper.status_extended.get()
199     resp.status_string
200
201     # Chassis Traffic
202     await tester.traffic.set(on_off=enums.OnOff.ON, module_ports=[0,0,
    ↪0,1])
203     await tester.traffic.set(on_off=enums.OnOff.OFF, module_ports=[0,0,
    ↪0,1])
204     await tester.traffic.set_on(module_ports=[0,0,0,1])
205     await tester.traffic.set_off(module_ports=[0,0,0,1])
206
207     # Synchronized Chassis Traffic
208     await tester.traffic_sync.set(on_off=enums.OnOff.ON,␣
    ↪timestamp=1234567, module_ports=[0,0,0,1])
209     await tester.traffic_sync.set(on_off=enums.OnOff.OFF,␣
    ↪timestamp=1234567, module_ports=[0,0,0,1])
210     await tester.traffic_sync.set_on(timestamp=1234567, module_
    ↪ports=[0,0,0,1])
211     await tester.traffic_sync.set_off(timestamp=1234567, module_
```

(continues on next page)

(continued from previous page)

```
↪ports=[0,0,0,1])

# endregion

# region Module
    ###################################################
    #                    Module                       #
    ###################################################

    # Access module index 0 on the tester
    module = tester.modules.obtain(0)

    # Capabilities
    resp = await module.capabilities.get()
    resp.can_advanced_timing
    resp.can_local_time_adjust
    resp.can_media_config
    resp.can_ppm_sweep
    resp.can_tsn
    resp.is_chimera
    resp.max_clock_ppm

    # Name
    resp = await module.name.get()
    resp.name

    # Description
    await module.comment.set(comment="description")

    resp = await module.comment.get()
    resp.comment

    # Legacy Model
    resp = await module.model.get()
    resp.model

    # Model
    resp = await module.revision.get()
    resp.revision

    # Serial Number
    resp = await module.serial_number.get()
    resp.serial_number

    # Firmware Version
    resp = await module.version_number.get()
```

(continues on next page)

(continued from previous page)

```
    resp.version

    # Port Count
    resp = await module.port_count.get()
    resp.port_count

    # Status
    resp = await module.status.get()
    resp.temperature

    # Media Configuration
    await module.media.set(media_config=enums.MediaConfigurationType.BASE_T1)
    await module.media.set(media_config=enums.MediaConfigurationType.BASE_T1S)
    await module.media.set(media_config=enums.MediaConfigurationType.CFP)
    await module.media.set(media_config=enums.MediaConfigurationType.CFP4)
    await module.media.set(media_config=enums.MediaConfigurationType.CXP)
    await module.media.set(media_config=enums.MediaConfigurationType.OSFP800)
    await module.media.set(media_config=enums.MediaConfigurationType.OSFP800_ANLT)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFP112)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFP112_ANLT)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFP28_NRZ)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFP28_PAM4)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFP56_PAM4)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFPDD_NRZ)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFPDD_PAM4)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFPDD800)
    await module.media.set(media_config=enums.MediaConfigurationType.QSFPDD800_ANLT)
    await module.media.set(media_config=enums.MediaConfigurationType.SFP112)
    await module.media.set(media_config=enums.MediaConfigurationType.
```

(continues on next page)

(continued from previous page)

```
↪SFP28)
286     await module.media.set(media_config=enums.MediaConfigurationType.
↪SFP56)
287     await module.media.set(media_config=enums.MediaConfigurationType.
↪SFPDD)
288 
289     resp = await module.media.get()
290     resp.media_config
291 
292     # Supported Media
293     resp = await module.available_speeds.get()
294     resp.media_info_list
295 
296     # Port Configuration
297     await module.cfp.config.set(portspeed_list=[1, 800000])
298     await module.cfp.config.set(portspeed_list=[2, 400000, 400000])
299     await module.cfp.config.set(portspeed_list=[4, 200000, 200000,␣
↪200000, 200000])
300     await module.cfp.config.set(portspeed_list=[8, 100000, 100000,␣
↪100000, 100000, 100000, 100000, 100000, 100000])
301 
302     resp = await module.cfp.config.get()
303     resp.portspeed_list
304 
305     # Reservation
306     await module.reservation.set(operation=enums.ReservedAction.
↪RELEASE)
307     await module.reservation.set_release()
308     await module.reservation.set(operation=enums.ReservedAction.
↪RELINQUISH)
309     await module.reservation.set_relinquish()
310     await module.reservation.set(operation=enums.ReservedAction.
↪RESERVE)
311     await module.reservation.set_reserve()
312 
313     resp = await module.reservation.get()
314     resp.operation
315 
316     # Reserved By
317     resp = await module.reserved_by.get()
318     resp.username
319 
320     # TX Clock Filter Loop Bandwidth
321     if not isinstance(module, modules.ModuleChimera):
322         await module.advanced_timing.clock_tx.filter.set(filter_
↪bandwidth=enums.LoopBandwidth.BW103HZ)
```

(continues on next page)

(continued from previous page)

```
323         await module.advanced_timing.clock_tx.filter.set_bw103hz()
324         await module.advanced_timing.clock_tx.filter.set(filter_
    ↪bandwidth=enums.LoopBandwidth.BW1683HZ)
325         await module.advanced_timing.clock_tx.filter.set_bw1683hz()
326         await module.advanced_timing.clock_tx.filter.set(filter_
    ↪bandwidth=enums.LoopBandwidth.BW207HZ)
327         await module.advanced_timing.clock_tx.filter.set_bw207hz()
328         await module.advanced_timing.clock_tx.filter.set(filter_
    ↪bandwidth=enums.LoopBandwidth.BW416HZ)
329         await module.advanced_timing.clock_tx.filter.set_bw416hz()
330         await module.advanced_timing.clock_tx.filter.set(filter_
    ↪bandwidth=enums.LoopBandwidth.BW7019HZ)
331         await module.advanced_timing.clock_tx.filter.set_bw7019hz()
332
333         resp = await module.advanced_timing.clock_tx.filter.get()
334         resp.filter_bandwidth
335
336     # TX Clock Source
337     if not isinstance(module, modules.ModuleChimera):
338         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.MODULELOCALCLOCK)
339         await module.advanced_timing.clock_tx.source.set_
    ↪modulelocalclock()
340         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P0RXCLK)
341         await module.advanced_timing.clock_tx.source.set_p0rxclk()
342         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P1RXCLK)
343         await module.advanced_timing.clock_tx.source.set_p1rxclk()
344         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P2RXCLK)
345         await module.advanced_timing.clock_tx.source.set_p2rxclk()
346         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P3RXCLK)
347         await module.advanced_timing.clock_tx.source.set_p3rxclk()
348         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P4RXCLK)
349         await module.advanced_timing.clock_tx.source.set_p4rxclk()
350         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P5RXCLK)
351         await module.advanced_timing.clock_tx.source.set_p5rxclk()
352         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P6RXCLK)
353         await module.advanced_timing.clock_tx.source.set_p6rxclk()
354         await module.advanced_timing.clock_tx.source.set(tx_
    ↪clock=enums.TXClockSource.P7RXCLK)
```

(continues on next page)

(continued from previous page)

```
355         await module.advanced_timing.clock_tx.source.set_p7rxclk()
356
357         resp = await module.advanced_timing.clock_tx.source.get()
358         resp.tx_clock
359
360     # TX Clock Status
361     if not isinstance(module, modules.ModuleChimera):
362         resp = await module.advanced_timing.clock_tx.status.get()
363         resp.status
364
365     # SMA Status
366     if not isinstance(module, modules.ModuleChimera):
367         resp = await module.advanced_timing.sma.status.get()
368         resp.status
369
370     # SMA Input
371     if not isinstance(module, modules.ModuleChimera):
372         await module.advanced_timing.sma.input.set(sma_in=enums.
    ↪SMAInputFunction.NOT_USED)
373         await module.advanced_timing.sma.input.set_notused()
374         await module.advanced_timing.sma.input.set(sma_in=enums.
    ↪SMAInputFunction.TX10MHZ)
375         await module.advanced_timing.sma.input.set_tx10mhz()
376         await module.advanced_timing.sma.input.set(sma_in=enums.
    ↪SMAInputFunction.TX2MHZ)
377         await module.advanced_timing.sma.input.set_tx2mhz()
378
379         resp = await module.advanced_timing.sma.input.get()
380         resp.sma_in
381
382     # SMA Output
383     if not isinstance(module, modules.ModuleChimera):
384         await module.advanced_timing.sma.output.set(sma_out=enums.
    ↪SMAOutputFunction.DISABLED)
385         await module.advanced_timing.sma.output.set_disabled()
386         await module.advanced_timing.sma.output.set(sma_out=enums.
    ↪SMAOutputFunction.P0RXCLK)
387         await module.advanced_timing.sma.output.set_p0rxclk()
388         await module.advanced_timing.sma.output.set(sma_out=enums.
    ↪SMAOutputFunction.P0RXCLK2MHZ)
389         await module.advanced_timing.sma.output.set_p0rxclk2mhz()
390         await module.advanced_timing.sma.output.set(sma_out=enums.
    ↪SMAOutputFunction.P0SOF)
391         await module.advanced_timing.sma.output.set_p0sof()
392         await module.advanced_timing.sma.output.set(sma_out=enums.
    ↪SMAOutputFunction.P1RXCLK)
```

(continues on next page)

(continued from previous page)

```
        await module.advanced_timing.sma.output.set_p1rxclk()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P1RXCLK2MHZ)
        await module.advanced_timing.sma.output.set_p1rxclk2mhz()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P1SOF)
        await module.advanced_timing.sma.output.set_p1sof()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.PASSTHROUGH)
        await module.advanced_timing.sma.output.set_passthrough()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF10MHZ)
        await module.advanced_timing.sma.output.set_ref10mhz()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF125MHZ)
        await module.advanced_timing.sma.output.set_ref125mhz()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF156MHZ)
        await module.advanced_timing.sma.output.set_ref156mhz()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF2MHZ)
        await module.advanced_timing.sma.output.set_ref2mhz()
        await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.TS_PPS)
        await module.advanced_timing.sma.output.set_ts_pps()

        resp = await module.advanced_timing.sma.output.get()
        resp.sma_out

    # Local Clock Adjust
    await module.timing.clock_local_adjust.set(ppb=10)

    resp = await module.timing.clock_local_adjust.get()
    resp.ppb

    # Clock Sync Status
    resp = await module.timing.clock_sync_status.get()
    resp.m_clock_diff
    resp.m_correction
    resp.m_is_steady_state
    resp.m_tune_is_increase
    resp.m_tune_value

    # Clock Source
    await module.timing.source.set(source=enums.TimingSource.CHASSIS)
    await module.timing.source.set_chassis()
```

(continues on next page)

(continued from previous page)

```
    await module.timing.source.set(source=enums.TimingSource.EXTERNAL)
    await module.timing.source.set_external()
    await module.timing.source.set(source=enums.TimingSource.MODULE)
    await module.timing.source.set_module()

    resp = await module.timing.source.get()
    resp.source

    # Clock PPM Sweep Configuration
    FREYA_MODULES = (modules.MFreya800G4S1P_a, modules.MFreya800G4S1P_
↪b, modules.MFreya800G4S1POSFP_a, modules.MFreya800G4S1POSFP_b)
    if isinstance(module, FREYA_MODULES):
        await module.clock_sweep.config.set(mode=enums.PPMSweepMode.
↪OFF, ppb_step=10, step_delay=10, max_ppb=10, loops=1)
        await module.clock_sweep.config.set(mode=enums.PPMSweepMode.
↪TRIANGLE, ppb_step=10, step_delay=10, max_ppb=10, loops=1)

        resp = await module.clock_sweep.config.get()
        resp.mode
        resp.ppb_step
        resp.step_delay
        resp.max_ppb
        resp.loops

    # Clock PPM Sweep Status
    if isinstance(module, FREYA_MODULES):
        resp = await module.clock_sweep.status.get()
        resp.curr_step
        resp.curr_sweep
        resp.max_steps


    # Chimera - Bypass Mode
    if isinstance(module, modules.ModuleChimera):
        await module.emulator_bypass_mode.set(on_off=enums.OnOff.ON)
        await module.emulator_bypass_mode.set_on()
        await module.emulator_bypass_mode.set(on_off=enums.OnOff.OFF)
        await module.emulator_bypass_mode.set_off()

        resp = await module.emulator_bypass_mode.get()
        resp.on_off

    # Chimera - Latency Mode
    if isinstance(module, modules.ModuleChimera):
        await module.latency_mode.set(mode=enums.ImpairmentLatencyMode.
↪NORMAL)
```

(continues on next page)

(continued from previous page)

```
473         await module.latency_mode.set_normal()
474         await module.latency_mode.set(mode=enums.ImpairmentLatencyMode.
    ↪EXTENDED)
475         await module.latency_mode.set_extended()
476 
477         resp = await module.latency_mode.get()
478         resp.mode
479 
480     # Chimera - TX Clock Source
481     if isinstance(module, modules.ModuleChimera):
482         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪MODULELOCALCLOCK)
483         await module.tx_clock.source.set_modulelocalclock()
484         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P0RXCLK)
485         await module.tx_clock.source.set_p0rxclk()
486         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P1RXCLK)
487         await module.tx_clock.source.set_p1rxclk()
488         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P2RXCLK)
489         await module.tx_clock.source.set_p2rxclk()
490         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P3RXCLK)
491         await module.tx_clock.source.set_p3rxclk()
492         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P4RXCLK)
493         await module.tx_clock.source.set_p4rxclk()
494         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P5RXCLK)
495         await module.tx_clock.source.set_p5rxclk()
496         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P6RXCLK)
497         await module.tx_clock.source.set_p6rxclk()
498         await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
    ↪P7RXCLK)
499         await module.tx_clock.source.set_p7rxclk()
500 
501         resp = await module.tx_clock.source.get()
502         resp.tx_clock
503 
504     # Chimera - TX Clock Status
505     if isinstance(module, modules.ModuleChimera):
506         resp = await module.tx_clock.status.get()
507         resp.status
508 
```

(continues on next page)

(continued from previous page)

```
# endregion

# region Port
    ###################################################
    #                     Port                        #
    ###################################################

    port = module.ports.obtain(0)

    # Reset
    await port.reset.set()

    if isinstance(port, ports.PortChimera):
        return

    # Flash
    await port.flash.set(on_off=enums.OnOff.ON)
    await port.flash.set_on()
    await port.flash.set(on_off=enums.OnOff.OFF)
    await port.flash.set_off()

    resp = await port.flash.get()
    resp.on_off


    # MAC Address
    await port.net_config.mac_address.set(mac_address=Hex("000000000000
↪"))

    resp = await port.net_config.mac_address.get()
    resp.mac_address

    # IPv4 Address
    await port.net_config.ipv4.address.set(
        ipv4_address=ipaddress.IPv4Address("10.10.10.10"),
        subnet_mask=ipaddress.IPv4Address("255.255.255.0"),
        gateway=ipaddress.IPv4Address("10.10.1.1"),
        wild=ipaddress.IPv4Address("0.0.0.0"))

    resp = await port.net_config.ipv4.address.get()
    resp.ipv4_address
    resp.gateway
    resp.subnet_mask
    resp.wild

    # ARP Reply
```

(continues on next page)

(continued from previous page)

```
    await port.net_config.ipv4.arp_reply.set(on_off=enums.OnOff.ON)
    await port.net_config.ipv4.arp_reply.set(on_off=enums.OnOff.OFF)

    resp = await port.net_config.ipv4.arp_reply.get()
    resp.on_off

    # Ping Reply
    await port.net_config.ipv4.ping_reply.set(on_off=enums.OnOff.ON)
    await port.net_config.ipv4.ping_reply.set(on_off=enums.OnOff.OFF)

    resp = await port.net_config.ipv4.ping_reply.get()
    resp.on_off

    # IPv6 Address
    await port.net_config.ipv6.address.set(
        ipv6_address=ipaddress.IPv6Address("fc00::0002"),
        gateway=ipaddress.IPv6Address("fc00::0001"),
        subnet_prefix=7,
        wildcard_prefix=0
    )

    resp = await port.net_config.ipv6.address.get()
    resp.ipv6_address
    resp.gateway
    resp.subnet_prefix
    resp.wildcard_prefix

    # NDP Reply
    await port.net_config.ipv6.arp_reply.set(on_off=enums.OnOff.ON)
    await port.net_config.ipv6.arp_reply.set(on_off=enums.OnOff.OFF)

    resp = await port.net_config.ipv6.arp_reply.get()
    resp.on_off

    # IPv6 Ping Reply
    await port.net_config.ipv6.ping_reply.set(on_off=enums.OnOff.ON)
    await port.net_config.ipv6.ping_reply.set(on_off=enums.OnOff.OFF)

    resp = await port.net_config.ipv6.ping_reply.get()
    resp.on_off

    # ARP Table, https://github.com/xenanetworks/open-automation-
↪script-library/tree/main/ip_streams_arp_table
    await port.arp_rx_table.set(chunks=[])

    resp = await port.arp_rx_table.get()
```

(continues on next page)

(continued from previous page)

```
    resp.chunks

    # NDP Table, https://github.com/xenanetworks/open-automation-script-library/tree/main/ip_streams_arp_table
    await port.ndp_rx_table.set(chunks=[])

    resp = await port.ndp_rx_table.get()
    resp.chunks

    # Capture Trigger Criteria, https://github.com/xenanetworks/open-automation-script-library/blob/main/packet_capture/packet_capture.py
    await port.capturer.trigger.set(start_criteria=enums.StartTrigger.ON, start_criteria_filter=0, stop_criteria=enums.StopTrigger.FULL, stop_criteria_filter=0)
    await port.capturer.trigger.set(start_criteria=enums.StartTrigger.ON, start_criteria_filter=0, stop_criteria=enums.StopTrigger.USERSTOP, stop_criteria_filter=0)
    await port.capturer.trigger.set(start_criteria=enums.StartTrigger.FCSERR, start_criteria_filter=0, stop_criteria=enums.StopTrigger.FCSERR, stop_criteria_filter=0)
    await port.capturer.trigger.set(start_criteria=enums.StartTrigger.PLDERR, start_criteria_filter=0, stop_criteria=enums.StopTrigger.PLDERR, stop_criteria_filter=0)
    await port.capturer.trigger.set(start_criteria=enums.StartTrigger.FILTER, start_criteria_filter=0, stop_criteria=enums.StopTrigger.FILTER, stop_criteria_filter=0)

    resp = await port.capturer.trigger.get()
    resp.start_criteria
    resp.start_criteria_filter
    resp.stop_criteria
    resp.stop_criteria_filter

    # Capture - Frame to Keep, https://github.com/xenanetworks/open-automation-script-library/blob/main/packet_capture/packet_capture.py
    await port.capturer.keep.set(kind=enums.PacketType.ALL, index=0, byte_count=0)
    await port.capturer.keep.set_all()
    await port.capturer.keep.set(kind=enums.PacketType.FCSERR, index=0, byte_count=0)
    await port.capturer.keep.set_fcserr()
    await port.capturer.keep.set(kind=enums.PacketType.FILTER, index=0, byte_count=0)
    await port.capturer.keep.set_filter()
    await port.capturer.keep.set(kind=enums.PacketType.NOTPLD, index=0, byte_count=0)
```

(continues on next page)

(continued from previous page)

```
    await port.capturer.keep.set_notpld()
    await port.capturer.keep.set(kind=enums.PacketType.PLDERR, index=0,
↪ byte_count=0)
    await port.capturer.keep.set_plderr()
    await port.capturer.keep.set(kind=enums.PacketType.TPLD, index=0,␣
↪byte_count=0)
    await port.capturer.keep.set_tpld()

    resp = await port.capturer.keep.get()
    resp.kind
    resp.index
    resp.byte_count

    # Capture - State
    await port.capturer.state.set(on_off=enums.StartOrStop.START)
    await port.capturer.state.set_start()
    await port.capturer.state.set(on_off=enums.StartOrStop.STOP)
    await port.capturer.state.set_stop()

    resp = await port.capturer.state.get()
    resp.on_off

    # Capture - Statistics
    resp = await port.capturer.stats.get()
    resp.start_time
    resp.status

    # Read Captured Packets
    pkts = await port.capturer.obtain_captured()
    for i in range(len(pkts)):
        resp = await pkts[i].packet.get()
        print(f"Packet content # {i}: {resp.hex_data}")

    # Inter-frame Gap
    await port.interframe_gap.set(min_byte_count=20)

    resp = await port.interframe_gap.get()
    resp.min_byte_count

    # PAUSE Frames
    await port.pause.set(on_off=enums.OnOff.ON)
    await port.pause.set_on()
    await port.pause.set(on_off=enums.OnOff.OFF)
    await port.pause.set_off()

    resp = await port.pause.get()
```

(continues on next page)

(continued from previous page)

```
    resp.on_off

    # Auto-Train
    await port.autotrain.set(interval=1)

    resp = await port.autotrain.get()
    resp.interval

    # Gap Monitor
    await port.gap_monitor.set(start=100, stop=10)

    resp = await port.gap_monitor.get()
    resp.start
    resp.stop

    # Priority Flow Control
    await port.pfc_enable.set(
        cos_0=enums.OnOff.ON,
        cos_1=enums.OnOff.OFF,
        cos_2=enums.OnOff.ON,
        cos_3=enums.OnOff.OFF,
        cos_4=enums.OnOff.ON,
        cos_5=enums.OnOff.OFF,
        cos_6=enums.OnOff.ON,
        cos_7=enums.OnOff.OFF,
        )

    resp = await port.pfc_enable.get()
    resp.cos_0
    resp.cos_1
    resp.cos_2
    resp.cos_3
    resp.cos_4
    resp.cos_5
    resp.cos_6
    resp.cos_7

    # Loopback
    await port.loop_back.set(mode=enums.LoopbackMode.L1RX2TX)
    await port.loop_back.set_l1rx2tx()
    await port.loop_back.set(mode=enums.LoopbackMode.L2RX2TX)
    await port.loop_back.set_l2rx2tx()
    await port.loop_back.set(mode=enums.LoopbackMode.L3RX2TX)
    await port.loop_back.set_l3rx2tx()
    await port.loop_back.set(mode=enums.LoopbackMode.NONE)
    await port.loop_back.set_none()
```

(continues on next page)

(continued from previous page)

```
    await port.loop_back.set(mode=enums.LoopbackMode.PORT2PORT)
    await port.loop_back.set_port2port()
    await port.loop_back.set(mode=enums.LoopbackMode.TXOFF2RX)
    await port.loop_back.set_txoff2rx()
    await port.loop_back.set(mode=enums.LoopbackMode.TXON2RX)
    await port.loop_back.set_txon2rx()

    resp = await port.loop_back.get()
    resp.mode

    # BRR Mode
    await port.brr_mode.set(mode=enums.BRRMode.MASTER)
    await port.brr_mode.set_master()
    await port.brr_mode.set(mode=enums.BRRMode.SLAVE)
    await port.brr_mode.set_slave()

    resp = await port.brr_mode.get()
    resp.mode

    # MDI/MDIX Mode
    await port.mdix_mode.set(mode=enums.MDIXMode.AUTO)
    await port.mdix_mode.set_auto()
    await port.mdix_mode.set(mode=enums.MDIXMode.MDI)
    await port.mdix_mode.set_mdi()
    await port.mdix_mode.set(mode=enums.MDIXMode.MDIX)
    await port.mdix_mode.set_mdix()

    resp = await port.mdix_mode.get()
    resp.mode

    # EEE- Capabilities
    resp = await port.eee.capabilities.get()
    resp.eee_capabilities

    # EEE - Partner Capabilities
    resp = await port.eee.partner_capabilities.get()
    resp.cap_1000base_t
    resp.cap_100base_kx
    resp.cap_10gbase_kr
    resp.cap_10gbase_kx4
    resp.cap_10gbase_t

    # EEE - Control
    await port.eee.enable.set(on_off=enums.OnOff.OFF)
    await port.eee.enable.set_off()
    await port.eee.enable.set(on_off=enums.OnOff.ON)
```

(continues on next page)

(continued from previous page)

```
764     await port.eee.enable.set_on()
765
766     resp = await port.eee.enable.get()
767     resp.on_off
768
769     # EEE - Low Power TX Mode
770     await port.eee.mode.set(on_off=enums.OnOff.ON)
771     await port.eee.mode.set_off()
772     await port.eee.mode.set(on_off=enums.OnOff.OFF)
773     await port.eee.mode.set_on()
774
775     resp = await port.eee.mode.get()
776     resp.on_off
777
778     # EEE - RX Power
779     resp = await port.eee.rx_power.get()
780     resp.channel_a
781     resp.channel_b
782     resp.channel_c
783     resp.channel_d
784
785     # EEE - SNR Margin
786     resp = await port.eee.snr_margin.get()
787     resp.channel_a
788     resp.channel_b
789     resp.channel_c
790     resp.channel_d
791
792     # EEE - Status
793     resp = await port.eee.status.get()
794     resp.link_up
795     resp.rxc
796     resp.rxh
797     resp.txc
798     resp.txh
799
800     # Fault - Signaling
801     await port.fault.signaling.set(fault_signaling=enums.
    ↪FaultSignaling.DISABLED)
802     await port.fault.signaling.set_disabled()
803     await port.fault.signaling.set(fault_signaling=enums.
    ↪FaultSignaling.FORCE_LOCAL)
804     await port.fault.signaling.set_force_local()
805     await port.fault.signaling.set(fault_signaling=enums.
    ↪FaultSignaling.FORCE_REMOTE)
806     await port.fault.signaling.set_force_remote()
```

(continues on next page)

(continued from previous page)

```
807     await port.fault.signaling.set(fault_signaling=enums.
↪FaultSignaling.NORMAL)
808     await port.fault.signaling.set_normal()
809 
810     resp = await port.fault.signaling.get()
811     resp.fault_signaling
812 
813     # Fault - Status
814     resp = await port.fault.status.get()
815     resp.local_fault_status
816     resp.remote_fault_status
817 
818     # Interface
819     resp = await port.interface.get()
820     resp.interface
821 
822     # Description
823     await port.comment.set(comment="description")
824 
825     resp = await port.comment.get()
826     resp.comment
827 
828     # Status
829     resp = await port.status.get()
830     resp.optical_power
831 
832     # Latency Mode
833     if not isinstance(port, ports.PortChimera):
834         await port.latency_config.mode.set(mode=enums.LatencyMode.
↪FIRST2FIRST)
835         await port.latency_config.mode.set_first2first()
836         await port.latency_config.mode.set(mode=enums.LatencyMode.
↪FIRST2LAST)
837         await port.latency_config.mode.set_first2last()
838         await port.latency_config.mode.set(mode=enums.LatencyMode.
↪LAST2FIRST)
839         await port.latency_config.mode.set_last2first()
840         await port.latency_config.mode.set(mode=enums.LatencyMode.
↪LAST2LAST)
841         await port.latency_config.mode.set_last2last()
842 
843         resp = await port.latency_config.mode.get()
844         resp.mode
845 
846     # Latency Offset
847     if not isinstance(port, ports.PortChimera):
```

(continues on next page)

(continued from previous page)

```
        await port.latency_config.offset.set(offset=5)

        resp = await port.latency_config.offset.get()
        resp.offset

    # Link Flap - Control
    await port.pcs_pma.link_flap.enable.set(on_off=enums.OnOff.ON)
    await port.pcs_pma.link_flap.enable.set_on()
    await port.pcs_pma.link_flap.enable.set(on_off=enums.OnOff.OFF)
    await port.pcs_pma.link_flap.enable.set_off()

    resp = await port.pcs_pma.link_flap.enable.get()
    resp.on_off

    # Link Flap - Configuration
    await port.pcs_pma.link_flap.params.set(duration=10, period=20,␣
↪repetition=0)

    resp = await port.pcs_pma.link_flap.params.get()
    resp.duration
    resp.period
    resp.repetition

    # Multicast Mode
    await port.multicast.mode.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastOperation.JOIN,
        second_count=10)
    await port.multicast.mode.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastOperation.JOIN,
        second_count=10)
    await port.multicast.mode.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastOperation.LEAVE,
        second_count=10)
    await port.multicast.mode.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastOperation.OFF,
        second_count=10)
    await port.multicast.mode.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastOperation.ON,
        second_count=10)

    resp = await port.multicast.mode.get()
```

(continues on next page)

(continued from previous page)

```
    resp.ipv4_multicast_addresses
    resp.operation
    resp.second_count

    # Multicast Extended Mode
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.EXCLUDE,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV3
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.INCLUDE,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV3
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.JOIN,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.LEAVE,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.LEAVE_TO_ALL,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.GENERAL_QUERY,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.GROUP_QUERY,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
```

(continues on next page)

(continued from previous page)

```
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.ON,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )
    await port.multicast.mode_extended.set(
        ipv4_multicast_addresses=[],
        operation=enums.MulticastExtOperation.OFF,
        second_count=10,
        igmp_version=enums.IGMPVersion.IGMPV2
    )

    resp = await port.multicast.mode_extended.get()
    resp.ipv4_multicast_addresses
    resp.operation
    resp.second_count
    resp.igmp_version

    # Multicast Source List
    await port.multicast.source_list.set(ipv4_addresses=[])

    resp = await port.multicast.source_list.get()
    resp.ipv4_addresses


    # Multicast Header
    await port.multicast.header.set(header_count=1, header_
↪format=enums.MulticastHeaderFormat.VLAN, tag=10, pcp=0, dei=0)
    await port.multicast.header.set(header_count=0, header_
↪format=enums.MulticastHeaderFormat.NOHDR, tag=10, pcp=0, dei=0)

    resp = await port.multicast.header.get()
    resp.header_count
    resp.header_format
    resp.tag
    resp.pcp
    resp.dei

    # Random Seed
    await port.random_seed.set(seed=1)

    resp = await port.random_seed.get()
    resp.seed

```

(continues on next page)

(continued from previous page)

```
    # Checksum Offset
    await port.checksum.set(offset=14)

    resp = await port.checksum.get()
    resp.offset

    # Maximum Header Length
    await port.max_header_length.set(max_header_length=56)

    resp = await port.max_header_length.get()
    resp.max_header_length

    # MIX Weights
    await port.mix.weights.set(
        weight_56_bytes:=0,
        weight_60_bytes:=0,
        weight_64_bytes:=70,
        weight_70_bytes:=15,
        weight_78_bytes:=15,
        weight_92_bytes:=0,
        weight_256_bytes:=0,
        weight_496_bytes:=0,
        weight_512_bytes:=0,
        weight_570_bytes:=0,
        weight_576_bytes:=0,
        weight_594_bytes:=0,
        weight_1438_bytes:=0,
        weight_1518_bytes:=0,
        weight_9216_bytes:=0,
        weight_16360_bytes:=0)

    resp = await port.mix.weights.get()
    resp.weight_56_bytes
    resp.weight_60_bytes
    resp.weight_64_bytes
    resp.weight_70_bytes
    resp.weight_78_bytes
    resp.weight_92_bytes
    resp.weight_256_bytes
    resp.weight_496_bytes
    resp.weight_512_bytes
    resp.weight_570_bytes
    resp.weight_576_bytes
    resp.weight_594_bytes
    resp.weight_1438_bytes
    resp.weight_1518_bytes
```

(continues on next page)

(continued from previous page)

```
    resp.weight_9216_bytes
    resp.weight_16360_bytes

    # MIX Lengths
    await port.mix.lengths[0].set(frame_size=56)
    await port.mix.lengths[1].set(frame_size=60)
    await port.mix.lengths[14].set(frame_size=9216)
    await port.mix.lengths[15].set(frame_size=16360)

    resp = await port.mix.lengths[0].get()
    resp.frame_size
    resp = await port.mix.lengths[1].get()
    resp.frame_size
    resp = await port.mix.lengths[14].get()
    resp.frame_size
    resp = await port.mix.lengths[15].get()
    resp.frame_size

    # Payload Mode
    await port.payload_mode.set(mode=enums.PayloadMode.NORMAL)
    await port.payload_mode.set_normal()
    await port.payload_mode.set(mode=enums.PayloadMode.EXTPL)
    await port.payload_mode.set_extpl()
    await port.payload_mode.set(mode=enums.PayloadMode.CDF)
    await port.payload_mode.set_cdf()

    resp = await port.payload_mode.get()
    resp.mode

    # RX Preamble Insert
    await port.preamble.rx_insert.set(on_off=enums.OnOff.ON)
    await port.preamble.rx_insert.set(on_off=enums.OnOff.OFF)

    resp = await port.preamble.rx_insert.get()
    resp.on_off

    # TX Preamble Removal
    await port.preamble.tx_remove.set(on_off=enums.OnOff.ON)
    await port.preamble.tx_remove.set(on_off=enums.OnOff.OFF)

    resp = await port.preamble.tx_remove.get()
    resp.on_off

    # Reservation
    await port.reservation.set(operation=enums.ReservedAction.RELEASE)
    await port.reservation.set_release()
```

(continues on next page)

(continued from previous page)

```
    await port.reservation.set(operation=enums.ReservedAction.
↪RELINQUISH)
    await port.reservation.set_relinquish()
    await port.reservation.set(operation=enums.ReservedAction.RESERVE)
    await port.reservation.set_reserve()

    resp = await port.reservation.get()
    resp.status

    # Reserved By
    resp = await port.reserved_by.get()
    resp.username

    # Runt - RX Length
    await port.runt.rx_length.set(runt_length=40)

    resp = await port.runt.rx_length.get()
    resp.runt_length

    # Runt - TX Length
    await port.runt.tx_length.set(runt_length=40)

    resp = await port.runt.tx_length.get()
    resp.runt_length

    # Runt - Length Error
    resp = await port.runt.has_length_errors.get()
    resp.status

    # Speed Mode Selection
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.AUTO)
    await port.speed.mode.selection.set_auto()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10M)
    await port.speed.mode.selection.set_f10m()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F10M100M)
    await port.speed.mode.selection.set_f10m100m()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F10MHDX)
    await port.speed.mode.selection.set_f10mhdx()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100M)
    await port.speed.mode.selection.set_f100m()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F100M1G)
    await port.speed.mode.selection.set_f100m1g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
```

(continues on next page)

(continued from previous page)

```
↪F100M1G10G)
    await port.speed.mode.selection.set_f100m1g10g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F100M1G2500M)
    await port.speed.mode.selection.set_f100m1g2500m()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F100MHDX)
    await port.speed.mode.selection.set_f100mhdx()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F1G)
    await port.speed.mode.selection.set_f1g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F2500M)
    await port.speed.mode.selection.set_f2500m()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F5G)
    await port.speed.mode.selection.set_f5g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10G)
    await port.speed.mode.selection.set_f10g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F40G)
    await port.speed.mode.selection.set_f40g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100G)
    await port.speed.mode.selection.set_f100g()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪UNKNOWN)
    await port.speed.mode.selection.set_unknown()
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F200G)
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F400G)
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F800G)
    await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F1600G)

    resp = await port.speed.mode.selection.get()
    resp.mode

    # Supported Speed Modes
    resp = await port.speed.mode.supported.get()
    resp.auto
    resp.f10M
    resp.f100M
    resp.f1G
    resp.f10G
    resp.f40G
    resp.f100G
    resp.f10MHDX
    resp.f100MHDX
    resp.f10M100M
    resp.f100M1G
```

(continues on next page)

(continued from previous page)

```
    resp.f100M1G10G
    resp.f2500M
    resp.f5G
    resp.f100M1G2500M
    resp.f25G
    resp.f50G
    resp.f200G
    resp.f400G
    resp.f800G
    resp.f1600G

    # Current Speed
    resp = await port.speed.current.get()
    resp.port_speed

    # Speed Reduction
    await port.speed.reduction.set(ppm=100)

    resp = await port.speed.reduction.get()
    resp.ppm

    # Sync Status
    resp = await port.sync_status.get()
    resp.sync_status == enums.SyncStatus.IN_SYNC
    resp.sync_status == enums.SyncStatus.NO_SYNC

    # Transceiver Status
    resp = await port.tcvr_status.get()
    resp.rx_loss_lane_0
    resp.rx_loss_lane_1
    resp.rx_loss_lane_2
    resp.rx_loss_lane_3

    # Transceiver Read & Write
    my_int = 1234
    await port.transceiver.access_rw(page_address=0, register_
↪address=0).set(value=hex(my_int)[2:])

    resp = await port.transceiver.access_rw(page_address=0, register_
↪address=0).get()
    resp.value
    my_int_resp = int(resp.value, 16)
    print(f"Returned value: {my_int_resp}")

    # Transceiver Sequential Read & Write
    await port.transceiver.access_rw_seq(page_address=0, register_
```

(continues on next page)

(continued from previous page)

```
↪address=0, byte_count=4).set(value=Hex("00FF00FF"))
1201
1202    resp = await port.transceiver.access_rw_seq(page_address=0,␣
↪register_address=0, byte_count=4).get()
1203    resp.value
1204
1205    # Transceiver MII
1206    await port.transceiver.access_mii(register_address=0).
↪set(value=Hex("00"))
1207
1208    resp = await port.transceiver.access_mii(register_address=0).get()
1209    resp.value
1210
1211    # Transceiver Temperature
1212    resp = await port.transceiver.access_temperature().get()
1213    resp.integral_part
1214    resp.fractional_part
1215
1216    # Transceiver RX Laser Power
1217    resp = await port.pcs_pma.transceiver.rx_laser_power.get()
1218    resp.nanowatts
1219
1220    # Transceiver TX Laser Power
1221    resp = await port.pcs_pma.transceiver.tx_laser_power.get()
1222    resp.nanowatts
1223
1224    # Traffic Control - Rate Percent
1225    await port.rate.fraction.set(port_rate_ppm=1_000_000)
1226
1227    resp = await port.rate.fraction.get()
1228    resp.port_rate_ppm
1229
1230    # Traffic Control - Rate L2 Bits Per Second
1231    await port.rate.l2_bps.set(port_rate_bps=1_000_000)
1232
1233    resp = await port.rate.l2_bps.get()
1234    resp.port_rate_bps
1235
1236    # Traffic Control - Rate Frames Per Second
1237    await port.rate.pps.set(port_rate_pps=10_000)
1238
1239    resp = await port.rate.pps.get()
1240    resp.port_rate_pps
1241
1242    # Traffic Control - Start and Stop
1243    await port.traffic.state.set(on_off=enums.StartOrStop.START)
```

(continues on next page)

(continued from previous page)

```
    await port.traffic.state.set_start()
    await port.traffic.state.set(on_off=enums.StartOrStop.STOP)
    await port.traffic.state.set_stop()

    resp = await port.traffic.state.get()
    resp.on_off

    # Traffic Control - Traffic Error
    resp = await port.traffic.error.get()
    resp.error

    # Traffic Control - Single Frame TX
    await port.tx_single_pkt.send.set(hex_data=Hex(
↪"000000000001020304050608000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))

    # Traffic Control - Single Frame Time
    resp = await port.tx_single_pkt.time.get()
    resp.nanoseconds

    # TPLD Mode
    await port.tpld_mode.set(mode=enums.TPLDMode.NORMAL)
    await port.tpld_mode.set_normal()
    await port.tpld_mode.set(mode=enums.TPLDMode.MICRO)
    await port.tpld_mode.set_micro()

    resp = await port.tpld_mode.get()
    resp.mode

    # TX Mode
    await port.tx_config.mode.set(mode=enums.TXMode.NORMAL)
    await port.tx_config.mode.set_normal()
    await port.tx_config.mode.set(mode=enums.TXMode.BURST)
    await port.tx_config.mode.set_burst()
    await port.tx_config.mode.set(mode=enums.TXMode.SEQUENTIAL)
    await port.tx_config.mode.set_sequential()
    await port.tx_config.mode.set(mode=enums.TXMode.STRICTUNIFORM)
    await port.tx_config.mode.set_strictuniform()

    resp = await port.tx_config.mode.get()
    resp.mode

    # Burst Period
    await port.tx_config.burst_period.set(burst_period=100)

    resp = await port.tx_config.burst_period.get()
    resp.burst_period
```

(continues on next page)

(continued from previous page)

```
    # TX Delay
    await port.tx_config.delay.set(delay_val=100)

    resp = await port.tx_config.delay.get()
    resp.delay_val

    # TX Enable
    await port.tx_config.enable.set(on_off=enums.OnOff.ON)
    await port.tx_config.enable.set(on_off=enums.OnOff.OFF)

    resp = await port.tx_config.enable.get()
    resp.on_off

    # Packet Limit
    await port.tx_config.packet_limit.set(packet_count_limit=1_000_000)

    resp = await port.tx_config.packet_limit.get()
    resp.packet_count_limit

    # Time Limit
    await port.tx_config.time_limit.set(microseconds=1_000_000)

    resp = await port.tx_config.time_limit.get()
    resp.microseconds

    # TX Time Elapsed
    resp = await port.tx_config.time.get()
    resp.microseconds

    # Prepare TX
    await port.tx_config.prepare.set()

    # Dynamic Traffic Rate
    await port.dynamic.set(on_off=enums.OnOff.OFF)
    await port.dynamic.set_off()
    await port.dynamic.set(on_off=enums.OnOff.ON)
    await port.dynamic.set_on()

    resp = await port.dynamic.get()
    resp.on_off

    await port.uat.mode.set(mode=enums.OnOff.ON, delay=500)
    await port.uat.mode.set(mode=enums.OnOff.OFF, delay=500)
    await port.uat.frame_loss_ratio.get()

```

(continues on next page)

(continued from previous page)

```
    ##################################################
    #                  Port Filter                   #
    ##################################################

    # Create and Obtain
    # Create a filter on the port, and obtain the filter object. The␣
↪filter index is automatically assigned by the port.
    filter = await port.filters.create()

    # Obtain One or Multiple
    filter = port.filters.obtain(position_idx=0)
    filter_list = port.filters.obtain_multiple(*[0,1,2])

    # Remove
    # Remove a filter on the port with an explicit filter index by the␣
↪index manager of the port.
    await port.filters.remove(position_idx=0)

    # Filter - Enable
    await filter.enable.set(on_off=enums.OnOff.ON)
    await filter.enable.set_on()
    await filter.enable.set(on_off=enums.OnOff.OFF)
    await filter.enable.set_off()

    resp = await filter.enable.get()
    resp.on_off

    # Filter - Description
    await filter.comment.set(comment="this is a comment")

    resp = await filter.comment.get()
    resp.comment

    # Filter - Condition
    await filter.condition.set(and_expression_0=0, and_not_expression_
↪0=0, and_expression_1=1, and_not_expression_1=0, and_expression_2=0,␣
↪and_expression_3=0)

    resp = await filter.condition.get()
    resp.and_expression_0
    resp.and_not_expression_0
    resp.and_expression_1
    resp.and_not_expression_1
    resp.and_expression_2
    resp.and_expression_3

```

(continues on next page)

(continued from previous page)

```
    # Filter - String Representation
    await filter.string.set(string_name="this is name")

    resp = await filter.string.get()
    resp.string_name

    ################################################
    #               Port Length Term               #
    ################################################

    # Create and Obtain
    # Create a length term on the port, and obtain the length term object. The length term index is automatically assigned by the port.
    length_term = await port.length_terms.create()

    # Obtain One or Multiple
    length_term = port.length_terms.obtain(key=0)
    length_term_list = port.length_terms.obtain_multiple(*[0,1,2])

    # Remove
    # Remove a length term on the port with an explicit length term index by the index manager of the port.
    await port.length_terms.remove(position_idx=0)

    ################################################
    #               Port Match Term                #
    ################################################

    # Create and Obtain
    # Create a match term on the port, and obtain the match term object. The match term index is automatically assigned by the port.
    match_term = await port.match_terms.create()

    # Obtain One or Multiple
    length_term = port.match_terms.obtain(key=0)
    length_term_list = port.match_terms.obtain_multiple(*[0,1,2])

    # Remove
    # Remove a match term on the port with an explicit match term index by the index manager of the port.
    await port.match_terms.remove(position_idx=0)

    ################################################
    #               Port Histogram                 #
    ################################################

```

(continues on next page)

(continued from previous page)

```
    # Create and Obtain
    # Create a histogram on the port, and obtain the histogram object. The histogram index is automatically assigned by the port.
    dataset = await port.datasets.create()

    # Obtain One or Multiple
    length_term = port.datasets.obtain(key=0)
    length_term_list = port.datasets.obtain_multiple(*[0,1,2])

    # Remove
    # Remove a histogram on the port with an explicit histogram index by the index manager of the port.
    await port.datasets.remove(position_idx=0)

    ##################################################
    #               Port PCS/PMA                     #
    ##################################################

    # Auto-Negotiation Settings
    resp = await port.pcs_pma.auto_neg.settings.get()
    resp.tec_ability
    resp.fec_capable
    resp.fec_requested
    resp.pause_mode

    # Auto-Negotiation Status
    resp = await port.pcs_pma.auto_neg.status.get()
    resp.mode
    resp.auto_state
    resp.tec_ability
    resp.fec_capable
    resp.fec_requested
    resp.fec
    resp.pause_mode

    # Auto-Negotiation Selection
    # Only applicable to RJ45 ports
    await port.autoneg_selection.set(on_off=enums.OnOff.ON)
    await port.autoneg_selection.set_on()
    await port.autoneg_selection.set(on_off=enums.OnOff.OFF)
    await port.autoneg_selection.set_off()

    resp = await port.autoneg_selection.get()
    resp.on_off

    # FEC Mode
```

(continues on next page)

(continued from previous page)

```
    await port.pcs_pma.phy.auto_neg.set(fec_mode=enums.OnOff.ON,
↪reserved_1=0, reserved_2=0, reserved_3=0, reserved_4=0)

    await port.fec_mode.set(mode=enums.FECMode.RS_FEC)
    await port.fec_mode.set(mode=enums.FECMode.RS_FEC_KP)
    await port.fec_mode.set(mode=enums.FECMode.RS_FEC_KR)
    await port.fec_mode.set(mode=enums.FECMode.FC_FEC)
    await port.fec_mode.set(mode=enums.FECMode.OFF)
    await port.fec_mode.set(mode=enums.FECMode.ON)

    resp = await port.fec_mode.get()
    resp.mode

    # Link Training Settings
    await port.pcs_pma.link_training.settings.set(
        mode=enums.LinkTrainingMode.DISABLED,
        pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
        nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
        nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
        timeout_mode=enums.TimeoutMode.DEFAULT)
    await port.pcs_pma.link_training.settings.set(
        mode=enums.LinkTrainingMode.STANDALONE,
        pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
        nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
        nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
        timeout_mode=enums.TimeoutMode.DEFAULT)
    await port.pcs_pma.link_training.settings.set(
        mode=enums.LinkTrainingMode.INTERACTIVE,
        pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
        nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
        nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
        timeout_mode=enums.TimeoutMode.DISABLED)
    await port.pcs_pma.link_training.settings.set(
        mode=enums.LinkTrainingMode.START_AFTER_AUTONEG,
        pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
        nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
        nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
        timeout_mode=enums.TimeoutMode.DEFAULT)

    resp = await port.pcs_pma.link_training.settings.get()
    resp.mode
    resp.pam4_frame_size
    resp.nrz_pam4_init_cond
    resp.nrz_preset
    resp.timeout_mode

```

(continues on next page)

(continued from previous page)

```
    # Link Training Serdes Status
    resp = await port.pcs_pma.link_training.per_lane_status[0].get()
    resp.mode
    resp.failure
    resp.status

    # PMA Pulse Error Inject Control
    await port.pcs_pma.pma_pulse_err_inj.enable.set(on_off=enums.OnOff.
↪ON)
    await port.pcs_pma.pma_pulse_err_inj.enable.set_on()
    await port.pcs_pma.pma_pulse_err_inj.enable.set(on_off=enums.OnOff.
↪OFF)
    await port.pcs_pma.pma_pulse_err_inj.enable.set_off()

    resp = await port.pcs_pma.pma_pulse_err_inj.enable.get()
    resp.on_off

    # PMA Pulse Error Inject Configuration
    await port.pcs_pma.pma_pulse_err_inj.params.set(duration=1000,␣
↪period=1000, repetition=10, coeff=5, exp=-5)

    resp = await port.pcs_pma.pma_pulse_err_inj.params.get()
    resp.duration
    resp.period
    resp.coeff
    resp.exp

    # RX Status - Lane Error Counters
    resp = await port.pcs_pma.lanes[0].rx_status.errors.get()
    resp.alignment_error_count
    resp.corrected_fec_error_count
    resp.header_error_count

    # RX Status - Lock Status
    resp = await port.pcs_pma.lanes[0].rx_status.lock.get()
    resp.align_lock
    resp.header_lock

    # RX Status - Lane Status
    resp = await port.pcs_pma.lanes[0].rx_status.status.get()
    resp.skew
    resp.virtual_lane

    # RX Status - Clear Counters
    await port.pcs_pma.rx.clear.set()

```

(continues on next page)

(continued from previous page)

```
    # RX Status - RX FEC Stats
    resp = await port.pcs_pma.rx.fec_status.get()
    resp.stats_type
    resp.data_count
    resp.stats

    # RX Status - RX Total Stats
    resp = await port.pcs_pma.rx.total_status.get()
    resp.total_corrected_codeword_count
    resp.total_corrected_symbol_count
    resp.total_post_fec_ber
    resp.total_pre_fec_ber
    resp.total_rx_bit_count
    resp.total_rx_codeword_count
    resp.total_uncorrectable_codeword_count

    # TX Configuration - Error Counters
    resp = await port.pcs_pma.alarms.errors.get()
    resp.total_alarms
    resp.los_error_count
    resp.total_align_error_count
    resp.total_bip_error_count
    resp.total_fec_error_count
    resp.total_header_error_count
    resp.total_higher_error_count
    resp.total_pcs_error_count
    resp.valid_mask

    # TX Configuration - Error Generation Rate
    resp = await port.pcs_pma.error_gen.error_rate.get()
    resp.rate

    # TX Configuration - Error Generation Inject
    await port.pcs_pma.error_gen.inject_one.set()

    # TX Configuration - Error Injection
    await port.pcs_pma.lanes[0].tx_error_inject.set_alignerror()
    await port.pcs_pma.lanes[0].tx_error_inject.set_bip8error()
    await port.pcs_pma.lanes[0].tx_error_inject.set_headererror()

    # TX Configuration - Lane Configuration
    await port.pcs_pma.lanes[0].tx_config.set(virt_lane_index=1,␣
↪skew=10)

    resp = await port.pcs_pma.lanes[0].tx_config.get()
    resp.virt_lane_index
```

(continues on next page)

(continued from previous page)

```
    resp.skew

    ################################################
    #                Port Medium                   #
    ################################################

    # Eye Diagram Information
    resp = await port.serdes[0].eye_diagram.info.get()
    resp.width_mui
    resp.height_mv
    resp.h_slope_left
    resp.h_slope_right
    resp.y_intercept_left
    resp.y_intercept_right
    resp.r_squared_fit_left
    resp.r_squared_fit_right
    resp.est_rj_rms_left
    resp.est_rj_rms_right
    resp.est_dj_pp
    resp.v_slope_bottom
    resp.v_slope_top
    resp.x_intercept_bottom
    resp.x_intercept_top
    resp.r_squared_fit_bottom
    resp.r_squared_fit_top
    resp.est_rj_rms_bottom
    resp.est_rj_rms_top

    # Eye Diagram Bit Error Rate
    resp = await port.serdes[0].eye_diagram.ber.get()
    resp.eye_ber_estimation

    # Eye Diagram Dwell Bits
    resp = await port.serdes[0].eye_diagram.dwell_bits.get()
    resp.max_dwell_bit_count
    resp.min_dwell_bit_count

    # Eye Diagram Measure
    resp = await port.serdes[0].eye_diagram.measure.get()
    resp.status

    # Eye Diagram Resolution
    resp = await port.serdes[0].eye_diagram.resolution.get()
    resp.x_resolution
    resp.y_resolution

```

(continues on next page)

(continued from previous page)

```
    # Eye Diagram Data Columns
    resp = await port.serdes[0].eye_diagram.read_column[0].get()
    resp.valid_column_count
    resp.values
    resp.x_resolution
    resp.y_resolution

    # PHY - Signal Status
    resp = await port.pcs_pma.phy.signal_status.get()
    resp.phy_signal_status

    # PHY - Settings
    await port.pcs_pma.phy.settings.set(
        link_training_on_off=enums.OnOff.ON,
        precode_on_off=enums.OnOffDefault.DEFAULT,
        graycode_on_off=enums.OnOff.OFF, pam4_msb_lsb_swap=enums.OnOff.
↪OFF)

    resp = await port.pcs_pma.phy.settings.get()
    resp.link_training_on_off
    resp.precode_on_off
    resp.graycode_on_off
    resp.pam4_msb_lsb_swap

    # TX Tap Autotune
    await port.serdes[0].phy.autotune.set(on_off=enums.OnOff.ON)
    await port.serdes[0].phy.autotune.set_on()
    await port.serdes[0].phy.autotune.set(on_off=enums.OnOff.OFF)
    await port.serdes[0].phy.autotune.set_off()

    resp = await port.serdes[0].phy.autotune.get()
    resp.on_off

    # TX Tap Retune
    await port.serdes[0].phy.retune.set(dummy=1)

    # TX Tap Configuration
    await port.serdes[0].phy.tx_equalizer.set(pre2=0, pre1=0, main=86,␣
↪post1=0, post2=0, post3=0)
    resp = await port.serdes[0].phy.tx_equalizer.get()
    resp.pre2
    resp.pre
    resp.main
    resp.post
    resp.pre3_post2
    resp.post3
```

(continues on next page)

(continued from previous page)

```
    # RX Tap Configuration
    await port.serdes[0].phy.rx_equalizer.set(auto=0, ctle=0,␣
↪reserved=0)

    resp = await port.serdes[0].phy.rx_equalizer.get()
    resp.auto
    resp.ctle

    ##################################################
    #                Port PRBS                       #
    ##################################################

    # PRBS Configuration
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS7,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.CAUI_VIRTUAL,
        polynomial=enums.PRBSPolynomial.PRBS9,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.PERSECOND)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS10,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS11,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS13,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS15,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
```

(continues on next page)

(continued from previous page)

```
        polynomial=enums.PRBSPolynomial.PRBS20,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS23,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS31,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS49,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
    await port.serdes[0].prbs.config.type.set(
        prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
        polynomial=enums.PRBSPolynomial.PRBS58,
        invert=enums.PRBSInvertState.NON_INVERTED,
        statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)

    resp = await port.serdes[0].prbs.config.type.get()
    resp.prbs_inserted_type
    resp.polynomial
    resp.invert
    resp.statistics_mode


    # PRBS Statistics
    resp = await port.serdes[0].prbs.status.get()
    resp.byte_count
    resp.error_count
    resp.lock
# endregion

# region Statistics
    ###################################################
    #               Statistics                        #
    ###################################################

    # Error Counter
    resp = await port.errors_count.get()
    resp.error_count
```

(continues on next page)

(continued from previous page)

```
    # RX Statistics - Clear Counter
    await port.statistics.rx.clear.set()

    # RX Statistics - Calibrate
    await port.statistics.rx.calibrate.set()

    # RX Statistics - Total Counter
    resp = await port.statistics.rx.total.get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # RX Statistics - Non-TPLD Counter
    resp = await port.statistics.rx.no_tpld.get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # RX Statistics - PFC Counter
    resp = await port.statistics.rx.pfc_stats.get()
    resp.packet_count
    resp.quanta_pri_0
    resp.quanta_pri_1
    resp.quanta_pri_2
    resp.quanta_pri_3
    resp.quanta_pri_4
    resp.quanta_pri_5
    resp.quanta_pri_6
    resp.quanta_pri_7

    # RX Statistics - Extra Counter
    resp = await port.statistics.rx.extra.get()
    resp.fcs_error_count
    resp.pause_frame_count
    resp.gap_count
    resp.gap_duration
    resp.pause_frame_count
    resp.rx_arp_reply_count
    resp.rx_arp_request_count
    resp.rx_ping_reply_count
    resp.rx_ping_request_count

    # RX Statistics - Received TPLDs
```

(continues on next page)

(continued from previous page)

```
    await port.statistics.rx.obtain_available_tplds()

    # RX Statistics - TPLD - Error Counter
    resp = await port.statistics.rx.access_tpld(tpld_id=0).errors.get()
    resp.non_incre_payload_packet_count
    resp.non_incre_seq_event_count
    resp.swapped_seq_misorder_event_count

    # RX Statistics - TPLD - Latency Counter
    resp = await port.statistics.rx.access_tpld(tpld_id=0).latency.
↪get()
    resp.avg_last_sec
    resp.max_last_sec
    resp.min_last_sec
    resp.avg_val
    resp.max_val
    resp.min_val

    # RX Statistics - TPLD - Jitter Counter
    resp = await port.statistics.rx.access_tpld(tpld_id=0).jitter.get()
    resp.avg_last_sec
    resp.max_last_sec
    resp.min_last_sec
    resp.avg_val
    resp.max_val
    resp.min_val

    # RX Statistics - TPLD - Traffic Counter
    resp = await port.statistics.rx.access_tpld(tpld_id=0).traffic.
↪get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # RX Statistics - Filter Statistics
    resp = await port.statistics.rx.obtain_filter_statistics(filter=0).
↪get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # TX Statistics - Clear Counter
    await port.statistics.tx.clear.set()

```

(continues on next page)

(continued from previous page)

```
    # TX Statistics - Total Counter
    resp = await port.statistics.tx.total.get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # TX Statistics - Non-TPLD Counter
    resp = await port.statistics.tx.no_tpld.get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec

    # TX Statistics - Extra Counter
    resp = await port.statistics.tx.extra.get()
    resp.tx_arp_req_count

    # TX Statistics - Stream Counter
    resp = await port.statistics.tx.obtain_from_stream(stream=0).get()
    resp.byte_count_since_cleared
    resp.packet_count_since_cleared
    resp.bit_count_last_sec
    resp.packet_count_last_sec
# endregion

# region Stream
    ###################################################
    #                    Stream                       #
    ###################################################

    # Create and Obtain
    # Create a stream on the port, and obtain the stream object. The␣
↪stream index is automatically assigned by the port.
    stream = await port.streams.create()

    # Obtain One or Multiple
    stream = port.streams.obtain(0)
    stream_list = port.streams.obtain_multiple(*[0,1,2])

    # Remove
    # Remove a stream on the port with an explicit stream index.
    await port.streams.remove(position_idx=0)

    # Description
    await stream.comment.set(comment="description")
```

(continues on next page)

(continued from previous page)

```
    resp = await stream.comment.get()
    resp.comment

    # Test Payload ID
    await stream.tpld_id.set(test_payload_identifier=0)

    resp = await stream.tpld_id.get()
    resp.test_payload_identifier

    # State
    await stream.enable.set(state=enums.OnOffWithSuppress.OFF)
    await stream.enable.set_off()
    await stream.enable.set(state=enums.OnOffWithSuppress.ON)
    await stream.enable.set_on()
    await stream.enable.set(state=enums.OnOffWithSuppress.SUPPRESS)
    await stream.enable.set_suppress()

    resp = await stream.enable.get()
    resp.state

    # Header Protocol Segment
    await stream.packet.header.protocol.set(segments=[
        enums.ProtocolOption.ETHERNET,
        enums.ProtocolOption.VLAN,
        enums.ProtocolOption.IP,
        enums.ProtocolOption.UDP,
    ])

    # ETHERNET = 1
    # """Ethernet II"""
    # VLAN = 2
    # """VLAN"""
    # ARP = 3
    # """Address Resolution Protocol"""
    # IP = 4
    # """IPv4"""
    # IPV6 = 5
    # """IPv6"""
    # UDP = 6
    # """User Datagram Protocol (w/o checksum)"""
    # TCP = 7
    # """Transmission Control Protocol (w/o checksum)"""
    # LLC = 8
    # """Logic Link Control"""
    # SNAP = 9
```

(continues on next page)

(continued from previous page)

```
    # """Subnetwork Access Protocol"""
    # GTP = 10
    # """GPRS Tunnelling Protocol"""
    # ICMP = 11
    # """Internet Control Message Protocol"""
    # RTP = 12
    # """Real-time Transport Protocol"""
    # RTCP = 13
    # """RTP Control Protocol"""
    # STP = 14
    # """Spanning Tree Protocol"""
    # SCTP = 15
    # """Stream Control Transmission Protocol"""
    # MACCTRL = 16
    # """MAC Control"""
    # MPLS = 17
    # """Multiprotocol Label Switching"""
    # PBBTAG = 18
    # """Provider Backbone Bridge tag"""
    # FCOE = 19
    # """Fibre Channel over Ethernet"""
    # FC = 20
    # """Fibre Channel"""
    # FCOETAIL = 21
    # """Fibre Channel over Ethernet (tail)"""
    # IGMPV3L0 = 22
    # """IGMPv3 Membership Query L=0"""
    # IGMPV3L1 = 23
    # """IGMPv3 Membership Query L=1"""
    # UDPCHECK = 24
    # """User Datagram Protocol (w/ checksum)"""
    # IGMPV2 = 25
    # """Internet Group Management Protocol v2"""
    # MPLS_TP_OAM = 26
    # """MPLS-TP, OAM Header"""
    # GRE_NOCHECK = 27
    # """Generic Routing Encapsulation (w/o checksum)"""
    # GRE_CHECK = 28
    # """Generic Routing Encapsulation (w/ checksum)"""
    # TCPCHECK = 29
    # """Transmission Control Protocol (w/ checksum)"""
    # GTPV1L0 = 30
    # """GTPv1 (no options), GPRS Tunneling Protocol v1"""
    # GTPV1L1 = 31
    # """GTPv1 (w/ options), GPRS Tunneling Protocol v1"""
    # GTPV2L0 = 32
```

(continues on next page)

(continued from previous page)

```
    # """GTPv2 (no options), GPRS Tunneling Protocol v2"""
    # GTPV2L1 = 33
    # """GTPv2 (w/ options), GPRS Tunneling Protocol v2"""
    # IGMPV1 = 34
    # """Internet Group Management Protocol v1"""
    # PWETHCTRL = 35
    # """PW Ethernet Control Word"""
    # VXLAN = 36
    # """Virtual eXtensible LAN"""
    # ETHERNET_8023 = 37
    # """Ethernet 802.3"""
    # NVGRE = 38
    # """Generic Routing Encapsulation (Network Virtualization)"""
    # DHCPV4 = 39
    # """Dynamic Host Configuration Protocol (IPv4)"""
    # GENEVE = 40
    # """Generic Network Virtualization Encapsulation"""

    resp = await stream.packet.header.protocol.get()
    resp.segments

    # Header Value
    await stream.packet.header.data.set(
        hex_data=Hex(
"00000000000004F4BC7FFE90810000000800450000002A000000007F113BC4000000000000000000000000
"))

    resp = await stream.packet.header.data.get()
    resp.hex_data

    # Packet Size
    await stream.packet.length.set(length_type=enums.LengthType.FIXED,
min_val=64, max_val=64)
    await stream.packet.length.set(length_type=enums.LengthType.
INCREMENTING, min_val=64, max_val=1500)
    await stream.packet.length.set(length_type=enums.LengthType.
BUTTERFLY, min_val=64, max_val=1500)
    await stream.packet.length.set(length_type=enums.LengthType.RANDOM,
 min_val=64, max_val=1500)
    await stream.packet.length.set(length_type=enums.LengthType.MIX,
min_val=64, max_val=64)

    resp = await stream.packet.length.get()
    resp.length_type
    resp.min_val
    resp.max_val
```

(continues on next page)

(continued from previous page)

```
    # Packet Auto Size
    await stream.packet.auto_adjust.set()

    # Payload Type
    # Pattern string in hex, min = 1 byte, max = 18 bytes
    await stream.payload.content.set(payload_type=enums.PayloadType.PATTERN, hex_data=Hex("000102030405060708090A0B0C0D0E0FDEAD"))
    await stream.payload.content.set(payload_type=enums.PayloadType.PATTERN, hex_data=Hex("F5"))

    # Patter string ignored for non-pattern types
    await stream.payload.content.set(payload_type=enums.PayloadType.INC16, hex_data=Hex("F5"))
    await stream.payload.content.set_inc_word("00")
    await stream.payload.content.set(payload_type=enums.PayloadType.INC8, hex_data=Hex("F5"))
    await stream.payload.content.set_inc_byte("00")
    await stream.payload.content.set(payload_type=enums.PayloadType.DEC8, hex_data=Hex("F5"))
    await stream.payload.content.set_dec_byte("00")
    await stream.payload.content.set(payload_type=enums.PayloadType.DEC16, hex_data=Hex("F5"))
    await stream.payload.content.set_dec_word("00")
    await stream.payload.content.set(payload_type=enums.PayloadType.PRBS, hex_data=Hex("F5"))
    await stream.payload.content.set_prbs("00")
    await stream.payload.content.set(payload_type=enums.PayloadType.RANDOM, hex_data=Hex("F5"))
    await stream.payload.content.set_random("00")

    resp = await stream.payload.content.get()
    resp.hex_data
    resp.payload_type

    # Extended Payload
    # Use await port.payload_mode.set_extpl() to set the port's payload mode to Extended Payload.
    await stream.payload.extended.set(hex_data=Hex("00110022FF"))

    resp = await stream.payload.extended.get()
    resp.hex_data

    # Rate Fraction
    await stream.rate.fraction.set(stream_rate_ppm=1_000_000)
```

(continues on next page)

(continued from previous page)

```
    resp = await stream.rate.fraction.get()
    resp.stream_rate_ppm

    # Packet Rate
    await stream.rate.pps.set(stream_rate_pps=1_000)

    resp = await stream.rate.pps.get()
    resp.stream_rate_pps

    # Bit Rate L2
    await stream.rate.l2bps.set(l2_bps=1_000_000)

    resp = await stream.rate.l2bps.get()
    resp.l2_bps

    # Packet Limit
    await stream.packet.limit.set(packet_count=1_000)

    resp = await stream.packet.limit.get()
    resp.packet_count

    # Burst Size and Density
    await stream.burst.burstiness.set(size=20, density=80)

    resp = await stream.burst.burstiness.get()
    resp.size
    resp.density

    # Inter Burst/Packet Gap
    await stream.burst.gap.set(inter_packet_gap=30, inter_burst_gap=30)

    resp = await stream.burst.gap.get()
    resp.inter_packet_gap
    resp.inter_burst_gap

    # Priority Flow
    await stream.priority_flow.set(cos=enums.PFCMode.ZERO)
    await stream.priority_flow.set(cos=enums.PFCMode.ONE)
    await stream.priority_flow.set(cos=enums.PFCMode.TWO)
    await stream.priority_flow.set(cos=enums.PFCMode.THREE)
    await stream.priority_flow.set(cos=enums.PFCMode.FOUR)
    await stream.priority_flow.set(cos=enums.PFCMode.FIVE)
    await stream.priority_flow.set(cos=enums.PFCMode.SIX)
    await stream.priority_flow.set(cos=enums.PFCMode.SEVEN)
    await stream.priority_flow.set(cos=enums.PFCMode.VLAN_PCP)

```

(continues on next page)

(continued from previous page)

```
    resp = await stream.priority_flow.get()
    resp.cos

    # IPv4 Gateway Address
    await stream.gateway.ipv4.set(gateway=ipaddress.IPv4Address("10.10.
↪10.1"))

    resp = await stream.gateway.ipv4.get()
    resp.gateway

    # IPv6 Gateway Address
    await stream.gateway.ipv6.set(gateway=ipaddress.IPv6Address("::0001
↪"))

    resp = await stream.gateway.ipv6.get()
    resp.gateway

    # ARP Resolve Peer Address
    # You need to make sure either the port has a correct gateway or␣
↪the stream has a correct destination IP address to ARP resolve the␣
↪MAC address.
    resp = await stream.request.arp.get()
    resp.mac_address

    # PING Check IP Peer
    # You need to make sure either the port has a correct gateway or␣
↪the stream has a correct destination IP address to ping.
    resp = await stream.request.ping.get()
    resp.delay
    resp.time_to_live

    # Custom Data Field
    # Use await port.payload_mode.set_cdf() to set the port's payload␣
↪mode to Custom Data Field.

    # Field Offset
    await stream.cdf.offset.set(offset=1)

    resp = await stream.cdf.offset.get()
    resp.offset

    # Byte Count
    await stream.cdf.count.set(cdf_count=1)

    resp = await stream.cdf.count.get()
    resp.cdf_count
```

(continues on next page)

(continued from previous page)

```
    ################################################
    #          Stream Modifier                     #
    ################################################

    # Create
    await stream.packet.header.modifiers.configure(number=1)

    # Clear
    await stream.packet.header.modifiers.clear()

    # Obtain
    # Must create modifiers before obtain.
    modifier = stream.packet.header.modifiers.obtain(idx=0)

    # Range
    await modifier.range.set(min_val=0, step=10, max_val=9)

    resp = await modifier.range.get()
    resp.min_val
    resp.max_val
    resp.step

    # Position, Action, Mask
    await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.INC, repetition=1)
    await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.DEC, repetition=1)
    await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.RANDOM, repetition=1)

    resp = await modifier.specification.get()
    resp.action
    resp.mask
    resp.position
    resp.repetition

    # ################################################
    # #           Stream 32-bit Modifier             #
    # ################################################

    # Create
    await stream.packet.header.modifiers_extended.configure(number=1)

    # Clear
    await stream.packet.header.modifiers_extended.clear()
```

(continues on next page)

(continued from previous page)

```
    # Obtain
    # Must create modifiers before obtain.
    modifier_ext = stream.packet.header.modifiers_extended.obtain(idx=0)

    # Range
    await modifier_ext.range.set(min_val=0, step=1, max_val=100)

    resp = await modifier_ext.range.get()
    resp.max_val
    resp.min_val
    resp.step

    # Position, Action, Mask
    await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"), action=enums.ModifierAction.INC, repetition=1)
    await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"), action=enums.ModifierAction.DEC, repetition=1)
    await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"), action=enums.ModifierAction.RANDOM, repetition=1)

    resp = await modifier_ext.specification.get()
    resp.action
    resp.mask
    resp.position
    resp.repetition

    # ##################################################
    # #            Stream Error Control                #
    # ##################################################

    # Misorder Error Injection
    await stream.inject_err.misorder.set()

    # Payload Integrity Error Injection
    await stream.inject_err.payload_integrity.set()

    # Sequence Error Injection
    await stream.inject_err.sequence.set()

    # Test Payload Error Injection
    await stream.inject_err.test_payload.set()

    # Checksum Error Injection
    await stream.inject_err.frame_checksum.set()
```

(continues on next page)

(continued from previous page)

```
2250
2251        # Insert Frame Checksum
2252        await stream.insert_packets_checksum.set(on_off=enums.OnOff.ON)
2253        await stream.insert_packets_checksum.set_on()
2254        await stream.insert_packets_checksum.set(on_off=enums.OnOff.OFF)
2255        await stream.insert_packets_checksum.set_off()
2256
2257        resp = await stream.insert_packets_checksum.get()
2258        resp.on_off
2259    # endregion
2260
2261    # region Network Emulation
2262        ################################################
2263        #               Network Emulation              #
2264        ################################################
2265
2266        # Configure Chimera port
2267        if isinstance(module, modules.ModuleChimera):
2268            port = module.ports.obtain(0)
2269
2270            await port.pcs_pma.link_flap.params.set(duration=100,␣
    ↪period=1000, repetition=0)
2271            await port.pcs_pma.link_flap.enable.set_on()
2272            await port.pcs_pma.link_flap.enable.set_off()
2273
2274            await port.pcs_pma.pma_pulse_err_inj.params.set(duration=100,␣
    ↪period=1000, repetition=0, coeff=100, exp=-4)
2275            await port.pcs_pma.pma_pulse_err_inj.enable.set_on()
2276            await port.pcs_pma.pma_pulse_err_inj.enable.set_off()
2277
2278            # Enable impairment on the port.
2279            await port.emulate.set_off()
2280            await port.emulate.set_on()
2281
2282            resp = await port.emulate.get()
2283            resp.action
2284
2285            # Set TPLD mode
2286            await port.emulation.tpld_mode.set(mode=enums.TPLDMode.NORMAL)
2287            await port.emulation.tpld_mode.set(mode=enums.TPLDMode.MICRO)
2288
2289            resp = await port.emulation.tpld_mode.get()
2290            resp.mode
2291
2292            # Configure flow's basic filter on a port
2293            # Configure flow properties
```

(continues on next page)

(continued from previous page)

```
        flow = port.emulation.flows[1]

        await flow.comment.set(comment="Flow description")

        resp = await flow.comment.get()
        resp.comment

        # Initializing the shadow copy of the filter.
        await flow.shadow_filter.initiating.set()

        # Configure shadow filter to BASIC mode
        await flow.shadow_filter.use_basic_mode()

        # Query the mode of the filter (either basic or extended)
        filter = await flow.shadow_filter.get_mode()

        if isinstance(filter, misc.BasicImpairmentFlowFilter):
            #------------------
            # Ethernet subfilter
            #------------------
            # Use and configure basic-mode shadow filter's Ethernet␣
↪subfilter
            await utils.apply(
                filter.ethernet.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
                filter.ethernet.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
                filter.ethernet.src_address.set(use=enums.OnOff.ON,␣
↪value=Hex("AAAAAAAAAAAA"), mask=Hex("FFFFFFFFFFFF")),
                filter.ethernet.dest_address.set(use=enums.OnOff.ON,␣
↪value=Hex("BBBBBBBBBBBB"), mask=Hex("FFFFFFFFFFFF"))
            )

            #------------------
            # Layer 2+ subfilter
            #------------------
            # Not use basic-mode shadow filter's Layer 2+ subfilter
            await filter.l2plus_use.set(use=enums.L2PlusPresent.NA)

            # Use and configure basic-mode shadow filter's Layer2+␣
↪subfilter (One VLAN tag)
            await utils.apply(
                filter.l2plus_use.set(use=enums.L2PlusPresent.VLAN1),
                filter.vlan.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
                filter.vlan.settings.set(use=enums.FilterUse.AND,␣
```

(continues on next page)

(continued from previous page)

```
↪action=enums.InfoAction.INCLUDE),
2333                filter.vlan.inner.tag.set(use=enums.OnOff.ON,␣
↪value=1234, mask=Hex("0FFF")),
2334                filter.vlan.inner.pcp.set(use=enums.OnOff.OFF, value=3,
↪ mask=Hex("07")),
2335            )
2336            # Use and configure basic-mode shadow filter's Layer2+␣
↪subfilter (Two VLAN tag)
2337            await utils.apply(
2338                filter.l2plus_use.set(use=enums.L2PlusPresent.VLAN2),
2339                filter.vlan.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2340                filter.vlan.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2341                filter.vlan.inner.tag.set(use=enums.OnOff.ON,␣
↪value=1234, mask=Hex("0FFF")),
2342                filter.vlan.inner.pcp.set(use=enums.OnOff.OFF, value=3,
↪ mask=Hex("07")),
2343                filter.vlan.outer.tag.set(use=enums.OnOff.ON,␣
↪value=2345, mask=Hex("0FFF")),
2344                filter.vlan.outer.pcp.set(use=enums.OnOff.OFF, value=0,
↪ mask=Hex("07")),
2345            )
2346            # Use and configure basic-mode shadow filter's Layer2+␣
↪subfilter (MPLS)
2347            await utils.apply(
2348                filter.l2plus_use.set(use=enums.L2PlusPresent.MPLS),
2349                filter.mpls.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2350                filter.mpls.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2351                filter.mpls.label.set(use=enums.OnOff.ON, value=1000,␣
↪mask=Hex("FFFFF")),
2352                filter.mpls.toc.set(use=enums.OnOff.ON, value=0,␣
↪mask=Hex("07")),
2353            )
2354
2355            #------------------
2356            # Layer 3 subfilter
2357            #------------------
2358            # Not use basic-mode shadow filter's Layer 3 subfilter
2359            await filter.l3_use.set(use=enums.L3Present.NA)
2360            # Use and configure basic-mode shadow filter's Layer 3␣
↪subfilter (IPv4)
2361            await utils.apply(
2362                filter.l3_use.set(use=enums.L3Present.IP4),
```

(continues on next page)

(continued from previous page)

```
2363                filter.ip.v4.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2364                filter.ip.v4.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2365                filter.ip.v4.src_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF")),
2366                filter.ip.v4.dest_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF")),
2367                filter.ip.v4.dscp.set(use=enums.OnOff.ON, value=0,␣
↪mask=Hex("FC")),
2368            )
2369            # Use and configure basic-mode shadow filter's Layer 3␣
↪subfilter (IPv6)
2370            await utils.apply(
2371                filter.l3_use.set(use=enums.L3Present.IP6),
2372                filter.ip.v6.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2373                filter.ip.v6.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2374                filter.ip.v6.src_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv6Address("2001::2"), mask=Hex(
↪"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF")),
2375                filter.ip.v6.dest_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv6Address("2002::2"), mask=Hex(
↪"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF")),
2376                filter.ip.v6.traffic_class.set(use=enums.OnOff.ON,␣
↪value=0, mask=Hex("FC")),
2377            )
2378
2379            #------------------
2380            # Layer 4 subfilter
2381            #------------------
2382            # Use and configure basic-mode shadow filter's Layer 4␣
↪subfilter (TCP)
2383            await utils.apply(
2384                filter.tcp.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2385                filter.tcp.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2386                filter.tcp.src_port.set(use=enums.OnOff.ON, value=1234,
↪ mask=Hex("FFFF")),
2387                filter.tcp.dest_port.set(use=enums.OnOff.ON, value=80,␣
↪mask=Hex("FFFF")),
2388            )
2389            # Use and configure basic-mode shadow filter's Layer 4␣
↪subfilter (UDP)
```

(continues on next page)

(continued from previous page)

```
2390            await utils.apply(
2391                filter.udp.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
2392                filter.udp.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE),
2393                filter.udp.src_port.set(use=enums.OnOff.ON, value=1234,
↪ mask=Hex("FFFF")),
2394                filter.udp.dest_port.set(use=enums.OnOff.ON, value=80,␣
↪mask=Hex("FFFF")),
2395            )
2396
2397            #------------------
2398            # Layer Xena subfilter
2399            #------------------
2400            await utils.apply(
2401                filter.tpld.settings.set(action=enums.InfoAction.
↪EXCLUDE),
2402                filter.tpld.settings.set(action=enums.InfoAction.
↪INCLUDE),
2403                filter.tpld.test_payload_filters_config[0].
↪set(use=enums.OnOff.ON, id = 2),
2404                filter.tpld.test_payload_filters_config[0].
↪set(use=enums.OnOff.OFF, id = 2),
2405                filter.tpld.test_payload_filters_config[1].
↪set(use=enums.OnOff.ON, id = 4),
2406                filter.tpld.test_payload_filters_config[1].
↪set(use=enums.OnOff.OFF, id = 4),
2407                filter.tpld.test_payload_filters_config[2].
↪set(use=enums.OnOff.ON, id = 6),
2408                filter.tpld.test_payload_filters_config[2].
↪set(use=enums.OnOff.OFF, id = 6),
2409                filter.tpld.test_payload_filters_config[3].
↪set(use=enums.OnOff.ON, id = 8),
2410                filter.tpld.test_payload_filters_config[3].
↪set(use=enums.OnOff.OFF, id = 8),
2411                filter.tpld.test_payload_filters_config[4].
↪set(use=enums.OnOff.ON, id = 10),
2412                filter.tpld.test_payload_filters_config[4].
↪set(use=enums.OnOff.OFF, id = 10),
2413                filter.tpld.test_payload_filters_config[5].
↪set(use=enums.OnOff.ON, id = 20),
2414                filter.tpld.test_payload_filters_config[5].
↪set(use=enums.OnOff.OFF, id = 20),
2415                filter.tpld.test_payload_filters_config[6].
↪set(use=enums.OnOff.ON, id = 40),
2416                filter.tpld.test_payload_filters_config[6].
```

(continues on next page)

(continued from previous page)

```
↪set(use=enums.OnOff.OFF, id = 40),
                filter.tpld.test_payload_filters_config[7].
↪set(use=enums.OnOff.ON, id = 60),
                filter.tpld.test_payload_filters_config[7].
↪set(use=enums.OnOff.OFF, id = 60),
                filter.tpld.test_payload_filters_config[8].
↪set(use=enums.OnOff.ON, id = 80),
                filter.tpld.test_payload_filters_config[8].
↪set(use=enums.OnOff.OFF, id = 80),
                filter.tpld.test_payload_filters_config[9].
↪set(use=enums.OnOff.ON, id = 100),
                filter.tpld.test_payload_filters_config[9].
↪set(use=enums.OnOff.OFF, id = 100),
                filter.tpld.test_payload_filters_config[10].
↪set(use=enums.OnOff.ON, id = 102),
                filter.tpld.test_payload_filters_config[10].
↪set(use=enums.OnOff.OFF, id = 102),
                filter.tpld.test_payload_filters_config[11].
↪set(use=enums.OnOff.ON, id = 104),
                filter.tpld.test_payload_filters_config[11].
↪set(use=enums.OnOff.OFF, id = 104),
                filter.tpld.test_payload_filters_config[12].
↪set(use=enums.OnOff.ON, id = 106),
                filter.tpld.test_payload_filters_config[12].
↪set(use=enums.OnOff.OFF, id = 106),
                filter.tpld.test_payload_filters_config[13].
↪set(use=enums.OnOff.ON, id = 108),
                filter.tpld.test_payload_filters_config[13].
↪set(use=enums.OnOff.OFF, id = 108),
                filter.tpld.test_payload_filters_config[14].
↪set(use=enums.OnOff.ON, id = 110),
                filter.tpld.test_payload_filters_config[14].
↪set(use=enums.OnOff.OFF, id = 110),
                filter.tpld.test_payload_filters_config[15].
↪set(use=enums.OnOff.ON, id = 200),
                filter.tpld.test_payload_filters_config[15].
↪set(use=enums.OnOff.OFF, id = 200),
            )

            #------------------
            # Layer Any subfilter
            #------------------
            await utils.apply(
                filter.any.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE),
                filter.any.settings.set(use=enums.FilterUse.AND,␣
```

(continues on next page)

(continued from previous page)

```
↪action=enums.InfoAction.INCLUDE),
2443                filter.any.config.set(position=0, value=Hex(
↪"112233445566"), mask=Hex("112233445566"))
2444            )
2445
2446        # Apply the filter so the configuration data in the shadow␣
↪copy is committed to the working copy automatically.
2447        await flow.shadow_filter.enable.set_off()
2448        await flow.shadow_filter.enable.set_on()
2449        await flow.shadow_filter.apply.set()
2450
2451        # Configure flow's extended filter on a port
2452        # Configure flow properties
2453        flow = port.emulation.flows[1]
2454        await flow.comment.set("Flow description")
2455
2456        # Initializing the shadow copy of the filter.
2457        await flow.shadow_filter.initiating.set()
2458
2459        # Configure shadow filter to EXTENDED mode
2460        await flow.shadow_filter.use_extended_mode()
2461
2462        # Query the mode of the filter (either basic or extended)
2463        filter = await flow.shadow_filter.get_mode()
2464
2465        if isinstance(filter, misc.ExtendedImpairmentFlowFilter):
2466
2467            await filter.use_segments(
2468                enums.ProtocolOption.VLAN
2469                )
2470            protocol_segments = await filter.get_protocol_segments()
2471            await protocol_segments[0].value.set(value=Hex(
↪"AAAAAAAAAAAABBBBBBBBBBBB8100"))
2472            await protocol_segments[0].mask.set(masks=Hex(
↪"00000000000000000000000000000000"))
2473            await protocol_segments[1].value.set(value=Hex("0064FFFF"))
2474            await protocol_segments[1].mask.set(masks=Hex("00000000"))
2475
2476            await protocol_segments[1].get()
2477
2478        # Configure impairment - Drop
2479        # Fixed Burst distribution for impairment Drop
2480        await utils.apply(
2481            flow.impairment_distribution.drop_type_config.fixed_burst.
↪set(burst_size=5),
2482            flow.impairment_distribution.drop_type_config.schedule.
```

(continues on next page)

(continued from previous page)

```
↪set(duration=1, period=5), #repeat (duration = 1, period = x)
2483            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=0), #one shot
2484        )
2485
2486        # Random Burst distribution for impairment Drop
2487        await utils.apply(
2488            flow.impairment_distribution.drop_type_config.random_burst.
↪set(minimum=1, maximum=10, probability=10_000),
2489            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=1), # repeat pattern
2490            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=0, period=0), #continuous
2491        )
2492
2493        # Fixed Rate distribution for impairment Drop
2494        await utils.apply(
2495            flow.impairment_distribution.drop_type_config.fixed_rate.
↪set(probability=10_000),
2496            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=1),# repeat pattern
2497            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=0, period=0), #continuous
2498        )
2499
2500        # Bit Error Rate distribution for impairment Drop
2501        await utils.apply(
2502            flow.impairment_distribution.drop_type_config.bit_error_
↪rate.set(coef=1, exp=1),
2503            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=1),# repeat pattern
2504            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=0, period=0), #continuous
2505        )
2506
2507        # Random Rate distribution for impairment Drop
2508        await utils.apply(
2509            flow.impairment_distribution.drop_type_config.random_rate.
↪set(probability=10_000),
2510            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=1),# repeat pattern
2511            flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=0, period=0), #continuous
2512        )
2513
2514        # Gilbert Elliot distribution for impairment Drop
```

(continues on next page)

(continued from previous page)

```
        await utils.apply(
            flow.impairment_distribution.drop_type_config.ge.set(good_state_prob=0, good_state_trans_prob=0, bad_state_prob=0, bad_state_trans_prob=0),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Uniform distribution for impairment Drop
        await utils.apply(
            flow.impairment_distribution.drop_type_config.uniform.set(minimum=1, maximum=1),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Gaussian distribution for impairment Drop
        await utils.apply(
            flow.impairment_distribution.drop_type_config.gaussian.set(mean=1, std_deviation=1),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Poisson distribution for impairment Drop
        await utils.apply(
            flow.impairment_distribution.drop_type_config.poisson.set(mean=9),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Gamma distribution for impairment Drop
        await utils.apply(
            flow.impairment_distribution.drop_type_config.gamma.set(shape=1, scale=1),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1), # repeat pattern
```

(continues on next page)

(continued from previous page)

```
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Custom distribution for impairment Drop
        data_x=[0, 1] * 256
        await port.custom_distributions.assign(0)
        await port.custom_distributions[0].comment.set(comment="Example Custom Distribution")
        await port.custom_distributions[0].definition.set(linear=enums.OnOff.OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_x)
        await utils.apply(
            flow.impairment_distribution.drop_type_config.custom.set(cust_id=0),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=1, period=1),
            flow.impairment_distribution.drop_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Set distribution and start impairment Drop
        await flow.impairment_distribution.drop_type_config.enable.set_on()
        await flow.impairment_distribution.drop_type_config.enable.set_off()

        # Configure impairment - Misordering

        # Fixed Burst distribution for impairment Misordering
        # dist = distributions.misordering.FixedBurst(burst_size=1)
        # dist.repeat(period=5)
        # dist.one_shot()

        # Fixed Burst distribution for impairment Drop
        await utils.apply(
            flow.impairment_distribution.misorder_type_config.fixed_burst.set(burst_size=5),
            flow.impairment_distribution.misorder_type_config.schedule.set(duration=1, period=5), #repeat
            flow.impairment_distribution.misorder_type_config.schedule.set(duration=1, period=0), #one shot
        )

        # Fixed Rate distribution for impairment Drop
        await utils.apply(
```

(continues on next page)

(continued from previous page)

```
            flow.impairment_distribution.misorder_type_config.fixed_
↪rate.set(probability=10_000),
            flow.impairment_distribution.misorder_type_config.schedule.
↪set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.misorder_type_config.schedule.
↪set(duration=0, period=0), #continuous
        )

        # Set distribution and start impairment Misordering
        await flow.misordering.set(depth=1)
        await flow.impairment_distribution.misorder_type_config.enable.
↪set_on()
        await flow.impairment_distribution.misorder_type_config.enable.
↪set_off()

        # Configure impairment - Latency & Jitter
        # Fixed Burst distribution for impairment Latency & Jitter
        await flow.impairment_distribution.latency_jitter_type_config.
↪constant_delay.set(delay=100)

        # Random Burst distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪accumulate_and_burst.set(delay=1300),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=1, period=1), #repeat (duration = 1, period =␣
↪x)
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=1, period=0), #one shot
        )

        # Step distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪step.set(low=1300, high=77000),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )
        await flow.impairment_distribution.corruption_type_config.off.
↪set()

        # Uniform distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪uniform.set(minimum=1, maximum=1),
            flow.impairment_distribution.latency_jitter_type_config.
```

(continues on next page)

(continued from previous page)

```
↪schedule.set(duration=0, period=0), #continuous
        )

        # Gaussian distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪gaussian.set(mean=1, std_deviation=1),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        resp = await flow.latency_range.get()
        resp.


        # Poisson distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪poisson.set(mean=1),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Gamma distribution for impairment Latency & Jitter
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪gamma.set(shape=1, scale=1),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Custom distribution for impairment Latency & Jitter
        data_x=[0, 1] * 256
        await port.custom_distributions.assign(0)
        await port.custom_distributions[0].comment.set(comment=
↪"Example Custom Distribution")
        await port.custom_distributions[0].definition.set(linear=enums.
↪OnOff.OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_
↪x=data_x)
        await utils.apply(
            flow.impairment_distribution.latency_jitter_type_config.
↪custom.set(cust_id=0),
            flow.impairment_distribution.latency_jitter_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

```

(continues on next page)

(continued from previous page)

```
        # Set distribution and start impairment Latency & Jitter
        await flow.impairment_distribution.latency_jitter_type_config.
↪enable.set_on()
        await flow.impairment_distribution.latency_jitter_type_config.
↪enable.set_off()

        # Configure impairment - Duplication

        # Fixed Burst distribution for impairment Duplication
        # dist.one_shot()
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.fixed_
↪burst.set(burst_size=1300),
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=1, period=1), #repeat (duration = 1, period =␣
↪x)
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=1, period=0), #one shot
        )

        # Random Burst distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.
↪random_burst.set(minimum=1, maximum=1, probability=10_0000),
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Fixed Rate distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.fixed_
↪rate.set(probability=10_000),
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Bit Error Rate distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.bit_
↪error_rate.set(coef=1, exp=1),
            flow.impairment_distribution.duplication_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
```

(continues on next page)

(continued from previous page)

```
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Random Rate distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.random_rate.set(probability=10_000),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Gilbert Elliot distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.ge.set(good_state_prob=0, good_state_trans_prob=0, bad_state_prob=0, bad_state_trans_prob=0),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Uniform distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.uniform.set(minimum=1, maximum=1),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Gaussian distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.gaussian.set(mean=1, std_deviation=1),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Poisson distribution for impairment Duplication
```

(continues on next page)

(continued from previous page)

```
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.poisson.set(mean=9),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Gamma distribution for impairment Duplication
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.gamma.set(shape=1, scale=1),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Custom distribution for impairment Duplication
        data_x=[0, 1] * 256
        await port.custom_distributions.assign(0)
        await port.custom_distributions[0].comment.set(comment="Example Custom Distribution")
        await port.custom_distributions[0].definition.set(linear=enums.OnOff.OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_x)
        await utils.apply(
            flow.impairment_distribution.duplication_type_config.custom.set(cust_id=0),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=1, period=1),
            flow.impairment_distribution.duplication_type_config.schedule.set(duration=0, period=0), #continuous
        )

        # Set distribution and start impairment Duplication
        await flow.impairment_distribution.duplication_type_config.enable.set_on()
        await flow.impairment_distribution.duplication_type_config.enable.set_off()

        # Configure impairment - Corruption

        # Fixed Burst distribution for impairment Corruption
        await utils.apply(
```

(continues on next page)

(continued from previous page)

```
2742            flow.impairment_distribution.corruption_type_config.fixed_
↪burst.set(burst_size=1300),
2743            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1), #repeat (duration = 1, period =␣
↪x)
2744            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=0), #one shot
2745        )
2746        # Random Burst distribution for impairment Corruption
2747        await utils.apply(
2748            flow.impairment_distribution.corruption_type_config.random_
↪burst.set(minimum=1, maximum=1, probability=10_0000),
2749            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
2750            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
2751        )
2752
2753        # Fixed Rate distribution for impairment Corruption
2754        await utils.apply(
2755            flow.impairment_distribution.corruption_type_config.fixed_
↪rate.set(probability=10_000),
2756            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1), # repeat pattern
2757            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
2758        )
2759
2760        # Bit Error Rate distribution for impairment Corruption
2761        await utils.apply(
2762            flow.impairment_distribution.corruption_type_config.bit_
↪error_rate.set(coef=1, exp=1),
2763            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
2764            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
2765        )
2766
2767        # Random Rate distribution for impairment Corruption
2768        await utils.apply(
2769            flow.impairment_distribution.corruption_type_config.random_
↪rate.set(probability=10_000),
2770            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
2771            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
```

(continues on next page)

(continued from previous page)

```
        )

        # Gilbert Elliot distribution for impairment Corruption
        await utils.apply(
            flow.impairment_distribution.corruption_type_config.ge.
↪set(good_state_prob=0, good_state_trans_prob=0, bad_state_prob=0,␣
↪bad_state_trans_prob=0),
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Uniform distribution for impairment Corruption
        await utils.apply(
            flow.impairment_distribution.corruption_type_config.
↪uniform.set(minimum=1, maximum=1),
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Gaussian distribution for impairment Corruption
        await utils.apply(
            flow.impairment_distribution.corruption_type_config.
↪gaussian.set(mean=1, std_deviation=1),
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),# repeat pattern
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Poisson distribution for impairment Corruption
        await utils.apply(
            flow.impairment_distribution.corruption_type_config.
↪poisson.set(mean=9),
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1), # repeat pattern
            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
        )

        # Gamma distribution for impairment Corruption
        await utils.apply(
            flow.impairment_distribution.corruption_type_config.gamma.
```

(continues on next page)

(continued from previous page)

```
↪set(shape=1, scale=1),
2805            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1), # repeat pattern
2806            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
2807        )
2808
2809
2810        # Custom distribution for impairment Corruption
2811        data_x=[0, 1] * 256
2812        await port.custom_distributions.assign(0)
2813        await port.custom_distributions[0].comment.set(comment=
↪"Example Custom Distribution")
2814        await port.custom_distributions[0].definition.set(linear=enums.
↪OnOff.OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_
↪x=data_x)
2815        await utils.apply(
2816            flow.impairment_distribution.corruption_type_config.custom.
↪set(cust_id=0),
2817            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=1, period=1),
2818            flow.impairment_distribution.corruption_type_config.
↪schedule.set(duration=0, period=0), #continuous
2819        )
2820
2821        # Set distribution and start impairment Corruption
2822        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪OFF)
2823        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪ETH)
2824        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪IP)
2825        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪TCP)
2826        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪UDP)
2827        await flow.corruption.set(corruption_type=enums.CorruptionType.
↪BER)
2828        await flow.impairment_distribution.corruption_type_config.
↪enable.set_on()
2829        await flow.impairment_distribution.corruption_type_config.
↪enable.set_off()
2830
2831        await flow.impairment_distribution.corruption_type_config.
↪enable.set_on()
2832
```

(continues on next page)

(continued from previous page)

```
        resp = await flow.impairment_distribution.corruption_type_config.one_shot_status.get()
        resp.one_shot_status

        # Configure bandwidth control - Policer

        await flow.bandwidth_control.policer.set(on_off=enums.OnOff.ON, mode=enums.PolicerMode.L1, cir=10_000, cbs=1_000)
        await flow.bandwidth_control.policer.set(on_off=enums.OnOff.ON, mode=enums.PolicerMode.L2, cir=10_000, cbs=1_000)


        # Configure bandwidth control - Shaper

        # Set and start bandwidth control Shaper
        await flow.bandwidth_control.shaper.set(on_off=enums.OnOff.ON, mode=enums.PolicerMode.L1, cir=10_000, cbs=1_000, buffer_size=1_000)
        await flow.bandwidth_control.shaper.set(on_off=enums.OnOff.ON, mode=enums.PolicerMode.L2, cir=10_000, cbs=1_000, buffer_size=1_000)

        # Flow statistics

        rx_total = await flow.statistics.rx.total.get()
        rx_total.byte_count
        rx_total.packet_count
        rx_total.l2_bps
        rx_total.pps

        tx_total = await flow.statistics.tx.total.get()
        tx_total.byte_count
        tx_total.packet_count
        tx_total.l2_bps
        tx_total.pps

        flow_drop_total = await flow.statistics.total.drop_packets.get()
        flow_drop_total.pkt_drop_count_total
        flow_drop_total.pkt_drop_count_programmed
        flow_drop_total.pkt_drop_count_bandwidth
        flow_drop_total.pkt_drop_count_other
        flow_drop_total.pkt_drop_ratio_total
        flow_drop_total.pkt_drop_ratio_programmed
        flow_drop_total.pkt_drop_ratio_bandwidth
        flow_drop_total.pkt_drop_ratio_other

        flow_corrupted_total = await flow.statistics.total.corrupted_
```

(continues on next page)

(continued from previous page)

```
↪packets.get()
        flow_corrupted_total.fcs_corrupted_pkt_count
        flow_corrupted_total.fcs_corrupted_pkt_ratio
        flow_corrupted_total.ip_corrupted_pkt_count
        flow_corrupted_total.ip_corrupted_pkt_ratio
        flow_corrupted_total.tcp_corrupted_pkt_count
        flow_corrupted_total.tcp_corrupted_pkt_ratio
        flow_corrupted_total.total_corrupted_pkt_count
        flow_corrupted_total.total_corrupted_pkt_ratio
        flow_corrupted_total.udp_corrupted_pkt_count
        flow_corrupted_total.udp_corrupted_pkt_ratio

        flow_delayed_total = await flow.statistics.total.latency_
↪packets.get()
        flow_delayed_total.pkt_count
        flow_delayed_total.ratio

        flow_jittered_total = await flow.statistics.total.jittered_
↪packets.get()
        flow_jittered_total.pkt_count
        flow_jittered_total.ratio

        flow_duplicated_total = await flow.statistics.total.duplicated_
↪packets.get()
        flow_duplicated_total.pkt_count
        flow_duplicated_total.ratio

        flow_misordered_total = await flow.statistics.total.mis_
↪ordered_packets.get()
        flow_misordered_total.pkt_count
        flow_misordered_total.ratio

        await flow.statistics.tx.clear.set()
        await flow.statistics.rx.clear.set()
        await flow.statistics.clear.set()

        # Port statistics
        port_drop = await port.emulation.statistics.drop.get()
        port_drop.pkt_drop_count_total
        port_drop.pkt_drop_count_programmed
        port_drop.pkt_drop_count_bandwidth
        port_drop.pkt_drop_count_other
        port_drop.pkt_drop_ratio_total
        port_drop.pkt_drop_ratio_programmed
        port_drop.pkt_drop_ratio_bandwidth
        port_drop.pkt_drop_ratio_other
```

(continues on next page)

(continued from previous page)

```
2914
2915        port_corrupted = await port.emulation.statistics.corrupted.
    ↪get()
2916        port_corrupted.fcs_corrupted_pkt_count
2917        port_corrupted.fcs_corrupted_pkt_ratio
2918        port_corrupted.ip_corrupted_pkt_count
2919        port_corrupted.ip_corrupted_pkt_ratio
2920        port_corrupted.tcp_corrupted_pkt_count
2921        port_corrupted.tcp_corrupted_pkt_ratio
2922        port_corrupted.total_corrupted_pkt_count
2923        port_corrupted.total_corrupted_pkt_ratio
2924        port_corrupted.udp_corrupted_pkt_count
2925        port_corrupted.udp_corrupted_pkt_ratio
2926
2927        port_delayed = await port.emulation.statistics.latency.get()
2928        port_delayed.pkt_count
2929        port_delayed.ratio
2930
2931        port_jittered = await port.emulation.statistics.jittered.get()
2932        port_jittered.pkt_count
2933        port_jittered.ratio
2934
2935        port_duplicated = await port.emulation.statistics.duplicated.
    ↪get()
2936        port_duplicated.pkt_count
2937        port_duplicated.ratio
2938
2939        port_misordered = await port.emulation.statistics.mis_ordered.
    ↪get()
2940        port_misordered.pkt_count
2941        port_misordered.ratio
2942
2943        await port.emulation.clear.set()
2944 # endregion
```

8.2.2 Tester

`xoa_driver.testers` includes tester APIs for Valkyrie, Vulcan, ValkyrieVE, and VulcanVE.

Action

Shutdown/Restart

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

Corresponding CLI command: C_DOWN

```
# Shutdown/Restart
await tester.down.set(operation=enums.ChassisShutdownAction.POWER_OFF)
await tester.down.set_poweroff()
await tester.down.set(operation=enums.ChassisShutdownAction.RESTART)
await tester.down.set_restart()
```

Flash

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.

Corresponding CLI command: C_FLASH

Note: Require Tester to be reserved before change value.

```
# Flash
await tester.flash.set(on_off=enums.OnOff.OFF)
await tester.flash.set_off()
await tester.flash.set(on_off=enums.OnOff.ON)
await tester.flash.set_on()

resp = await tester.flash.get()
resp.on_off
```

Debug Log

Allows to dump all the logs of a chassis.

Corresponding CLI command: C_DEBUGLOGS

```
# Debug Log
resp = await tester.debug_log.get()
resp.data
resp.message_length
```

Management Address

IP Address

The network configuration parameters of the chassis management port.

Corresponding CLI command: C_IPADDRESS

```
import ipaddress

# IP Address
await tester.management_interface.ip_address.set(
    ipv4_address=ipaddress.IPv4Address("10.10.10.10"),
    subnet_mask=ipaddress.IPv4Address("255.255.255.0"),
    gateway=ipaddress.IPv4Address("10.10.10.1"))

resp = await tester.management_interface.ip_address.get()
resp.ipv4_address
resp.subnet_mask
resp.gateway
```

MAC Address

Get the MAC address for the chassis management port.

Corresponding CLI command: C_MACADDRESS

```
# MAC Address
resp = await tester.management_interface.macaddress.get()
resp.mac_address
```

Hostname

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

Corresponding CLI command: C_HOSTNAME

```
# Hostname
await tester.management_interface.hostname.set(hostname="name")

resp = await tester.management_interface.hostname.get()
resp.hostname
```

DHCP

Controls whether the chassis will use DHCP to get the management IP address. Corresponding CLI command: C_DHCP

```
# DHCP
await tester.management_interface.dhcp.set(on_off=enums.OnOff.ON)
await tester.management_interface.dhcp.set_on()
await tester.management_interface.dhcp.set(on_off=enums.OnOff.OFF)
await tester.management_interface.dhcp.set_off()

resp = await tester.management_interface.dhcp.get()
resp.on_off
```

Capabilities

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

Corresponding CLI command: C_CAPABILITIES

```
# Capabilities
resp = await tester.capabilities.get()
resp.version
resp.max_name_len
resp.max_comment_len
resp.max_password_len
resp.max_ext_rate
resp.max_session_count
resp.max_chain_depth
resp.max_module_count
resp.max_protocol_count
resp.can_stream_based_arp
resp.can_sync_traffic_start
resp.can_read_log_files
resp.can_par_module_upgrade
resp.can_upgrade_timekeeper
resp.can_custom_defaults
resp.max_owner_name_length
resp.can_read_temperatures
resp.can_latency_f2f
```

Connection

Creating a test object automatically create a TCP connection to the tester.

Valkyrie

```
tester = await testers.L23Tester("0.0.0.0", "xoa")
```

Vulcan

```
tester = await testers.L47Tester("0.0.0.0", "xoa")
```

ValkyrieVE

```
tester = await testers.L23VeTester("0.0.0.0", "xoa")
```

VulcanVE

```
tester = await testers.L47VeTester("0.0.0.0", "xoa")
```

Identification

Name

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.

Corresponding CLI command: `C_NAME`

```
# Name
await tester.name.set(chassis_name="name")

resp = await tester.name.get()
resp.chassis_name
```

Password

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

Corresponding CLI command: `C_PASSWORD`

```
# Password
await tester.password.set(password="xena")

resp = await tester.password.get()
resp.password
```

Description

The description of the chassis.

Corresponding CLI command: `C_COMMENT`

```
# Description
await tester.comment.set(comment="description")

resp = await tester.comment.get()
resp.comment
```

Model

Gets the specific model of this Xena chassis.

Corresponding CLI command: `C_MODEL`

```
# Model
resp = await tester.model.get()
resp.model
```

Serial Number

Gets the unique serial number of this particular Xena chassis.

Corresponding CLI command: `C_SERIALNO`

```
# Serial Number
resp = await tester.serial_no.get()
resp.serial_number
```

Firmware Version

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

Corresponding CLI command: C_VERSIONNO

```
# Firmware Version
resp = await tester.version_no.get()
resp.chassis_major_version
resp.pci_driver_version

resp = await tester.version_no_minor.get()
resp.chassis_minor_version
resp.reserved_1
resp.reserved_2
```

Build String

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

Corresponding CLI command: C_BUILDSTRING

```
# Build String
resp = await tester.build_string.get()
resp.build_string
```

Reservation

Reservation Action

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>

Corresponding CLI command: C_RESERVATION

```
# Reservation
await tester.reservation.set(operation=enums.ReservedAction.RELEASE)
await tester.reservation.set_release()
await tester.reservation.set(operation=enums.ReservedAction.RELINQUISH)
```

(continues on next page)

(continued from previous page)

```
await tester.reservation.set_relinquish()
await tester.reservation.set(operation=enums.ReservedAction.RESERVE)
await tester.reservation.set_reserve()

resp = await tester.reservation.get()
resp.operation
```

Reserved By

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

Corresponding CLI command: `C_RESERVEDBY`

```
# Reserved By
resp = await tester.reserved_by.get()
resp.username
```

Session

Information

The following are pre-fetched in cache when connection is established, thus no need to use `await`.

```
tester.session.owner_name
tester.session.keepalive
tester.session.pwd
tester.session.is_online
tester.session.sessions_info
tester.session.timeout
tester.is_released()
tester.is_reserved_by_me()
```

Logoff

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

Corresponding CLI command: `C_LOGOFF`

```
await tester.session.logoff()
tester.on_disconnected(_callback_func)
```

Time and Timekeeper

Time

Get local chassis time in seconds.

Corresponding CLI command: C_TIME

```
# Time
resp = await tester.time.get()
resp.local_time
```

TimeKeeper Configuration

TimeKeeper config file content.

Corresponding CLI command: C_TKCONFIG

```
# TimeKeeper Configuration
await tester.time_keeper.config_file.set(config_file="filename")

resp = await tester.time_keeper.config_file.get()
resp.config_file
```

TimeKeeper GPS State

Get TimeKeeper GPS status.

Corresponding CLI command: C_TKGPSSTATE

```
# TimeKeeper GPS State
resp = await tester.time_keeper.gps_state.get()
resp.status
```

TimeKeeper License File

TimeKeeper license file content.

Corresponding CLI command: C_TKLICFILE

```
# TimeKeeper License File
await tester.time_keeper.license_file.set(license_content="")

resp = await tester.time_keeper.license_file.get()
resp.license_content
```

TimeKeeper License State

State of TimeKeeper license file content.

Corresponding CLI command: C_TKLICSTATE

```
# TimeKeeper License State
resp = await tester.time_keeper.license_state.get()
resp.license_errors
resp.license_file_state
resp.license_type
```

TimeKeeper Status

Version and status of TimeKeeper.

Corresponding CLI command: C_TKSTATUS

```
# TimeKeeper Status
resp = await tester.time_keeper.status.get()
resp.status_string

resp = await tester.time_keeper.status_extended.get()
resp.status_string
```

Traffic Control

Chassis Traffic

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

Corresponding CLI command: C_TRAFFIC

```
# Chassis Traffic
await tester.traffic.set(on_off=enums.OnOff.ON, module_ports=[0,0,0,1])
await tester.traffic.set(on_off=enums.OnOff.OFF, module_ports=[0,0,0,
↪1])
await tester.traffic.set_on(module_ports=[0,0,0,1])
await tester.traffic.set_off(module_ports=[0,0,0,1])
```

Synchronized Chassis Traffic

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.

Corresponding CLI command: C_TRAFFICSYNC

```
# Synchronized Chassis Traffic
await tester.traffic_sync.set(on_off=enums.OnOff.ON, timestamp=1234567,
↪ module_ports=[0,0,0,1])
await tester.traffic_sync.set(on_off=enums.OnOff.OFF,␣
↪timestamp=1234567, module_ports=[0,0,0,1])
await tester.traffic_sync.set_on(timestamp=1234567, module_ports=[0,0,
↪0,1])
await tester.traffic_sync.set_off(timestamp=1234567, module_ports=[0,0,
↪0,1])
```

8.2.3 Module

xoa_driver.modules includes module APIs for Valkyrie, Vulcan, Chimera, ValkyrieVE, and VulcanVE.

Capabilities

Gets the module capabilities.

Corresponding CLI command: M_CAPABILITIES

```
# Capabilities
resp = await module.capabilities.get()
resp.can_advanced_timing
resp.can_local_time_adjust
resp.can_media_config
resp.can_ppm_sweep
resp.can_tsn
resp.is_chimera
resp.max_clock_ppm
```

Identification

Name

Gets the name of a module.

Corresponding CLI command: `M_NAME`

```
# Name
resp = await module.name.get()
resp.name
```

Description

Gets the user-defined description string of a module.

Corresponding CLI command: `M_COMMENT`

```
# Description
await module.comment.set(comment="description")

resp = await module.comment.get()
resp.comment
```

Legacy Model

Gets the legacy model P/N name of a Xena test module.

Corresponding CLI command: `M_MODEL`

```
# Legacy Model
resp = await module.model.get()
resp.model
```

Model

Gets the model P/N name of a Xena test module.

Corresponding CLI command: `M_REVISION`

```
# Model
resp = await module.revision.get()
resp.revision
```

Serial Number

Gets the unique serial number of a module.

Corresponding CLI command: `M_SERIALNO`

```
# Serial Number
resp = await module.serial_number.get()
resp.serial_number
```

Firmware Version

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

Corresponding CLI command: `M_VERSIONNO`

```
# Firmware Version
resp = await module.version_number.get()
resp.version
```

Port Count

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_CFPCONFIGEXT command which returns the number of current ports.

Corresponding CLI command: `M_PORTCOUNT`

```
# Port Count
resp = await module.port_count.get()
resp.port_count
```

Status

Get status readings for the test module itself.

Corresponding CLI command: `M_STATUS`

```
# Status
resp = await module.status.get()
resp.temperature
```

Impairment

Note: For Chimera module only.

Bypass Mode

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

Corresponding CLI command: `M_EMULBYPASS`

```
# Chimera - Bypass Mode
if isinstance(module, modules.ModuleChimera):
    await module.emulator_bypass_mode.set(on_off=enums.OnOff.ON)
    await module.emulator_bypass_mode.set_on()
    await module.emulator_bypass_mode.set(on_off=enums.OnOff.OFF)
    await module.emulator_bypass_mode.set_off()

    resp = await module.emulator_bypass_mode.get()
    resp.on_off
```

Latency Mode

Configures the latency mode for 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.

Corresponding CLI command: `M_LATENCYMODE`

```
# Chimera - Latency Mode
if isinstance(module, modules.ModuleChimera):
    await module.latency_mode.set(mode=enums.ImpairmentLatencyMode.
↪NORMAL)
    await module.latency_mode.set_normal()
    await module.latency_mode.set(mode=enums.ImpairmentLatencyMode.
↪EXTENDED)
    await module.latency_mode.set_extended()

    resp = await module.latency_mode.get()
    resp.mode
```

TX Clock Source

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

Corresponding CLI command: `M_TXCLOCKSOURCE_NEW`

```
# Chimera - TX Clock Source
if isinstance(module, modules.ModuleChimera):
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪MODULELOCALCLOCK)
    await module.tx_clock.source.set_modulelocalclock()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P0RXCLK)
    await module.tx_clock.source.set_p0rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P1RXCLK)
    await module.tx_clock.source.set_p1rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P2RXCLK)
    await module.tx_clock.source.set_p2rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P3RXCLK)
    await module.tx_clock.source.set_p3rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P4RXCLK)
    await module.tx_clock.source.set_p4rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P5RXCLK)
    await module.tx_clock.source.set_p5rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P6RXCLK)
    await module.tx_clock.source.set_p6rxclk()
    await module.tx_clock.source.set(tx_clock=enums.TXClockSource.
↪P7RXCLK)
```

(continues on next page)

(continued from previous page)

```
    await module.tx_clock.source.set_p7rxclk()

    resp = await module.tx_clock.source.get()
    resp.tx_clock
```

TX Clock Status

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

Corresponding CLI command: `M_TXCLOCKSTATUS_NEW`

```
# Chimera - TX Clock Status
if isinstance(module, modules.ModuleChimera):
    resp = await module.tx_clock.status.get()
    resp.status
```

Media

Media Configuration

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

Corresponding CLI command: `M_MEDIA`

```
# Media Configuration
await module.media.set(media_config=enums.MediaConfigurationType.BASE_
↪T1)
await module.media.set(media_config=enums.MediaConfigurationType.BASE_
↪T1S)
await module.media.set(media_config=enums.MediaConfigurationType.CFP)
await module.media.set(media_config=enums.MediaConfigurationType.CFP4)
await module.media.set(media_config=enums.MediaConfigurationType.CXP)
await module.media.set(media_config=enums.MediaConfigurationType.
↪OSFP800)
await module.media.set(media_config=enums.MediaConfigurationType.
↪OSFP800_ANLT)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFP112)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFP112_ANLT)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFP28_NRZ)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFP28_PAM4)
```

(continues on next page)

(continued from previous page)

```
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFP56_PAM4)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFPDD_NRZ)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFPDD_PAM4)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFPDD800)
await module.media.set(media_config=enums.MediaConfigurationType.
↪QSFPDD800_ANLT)
await module.media.set(media_config=enums.MediaConfigurationType.
↪SFP112)
await module.media.set(media_config=enums.MediaConfigurationType.SFP28)
await module.media.set(media_config=enums.MediaConfigurationType.SFP56)
await module.media.set(media_config=enums.MediaConfigurationType.SFPDD)

resp = await module.media.get()
resp.media_config
```

Supported Media

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>] is repeated until all speeds supported by the <cage_type> has been listed. [<cage_type> <available_speed_count>] is repeated for all cage types on the module including the related <ports_per_speed> <speed> information.

Corresponding CLI command: M_MEDIASUPPORT

```
resp = await module.available_speeds.get()
resp.media_info_list
```

Port Configuration

This property defines the current number of ports and the speed of each of them on a CFP test module. The following combinations are possible: 2x10G, 4x10G, 8x10G, 2x25G, 4x25G, 8x25G, 1x40G, 2x40G, 2x50G, 4x50G, 8x50G, 1x100G, 2x100G, 4x100G, 2x200G, and 1x400G.

Note: <portspeed_list> is a list of integers, where the first element is the number of ports followed by a number of port speeds in Mbps.

The number of port speeds equals the value of the number of ports.

For example if the configuration is 4x25G, <portspeed_list> will be [4, 25000, 25000, 25000, 25000].

Corresponding CLI command: M_CFPCONFIGEXT

```
# Port Configuration
await module.cfp.config.set(portspeed_list=[1, 800000])
await module.cfp.config.set(portspeed_list=[2, 400000, 400000])
await module.cfp.config.set(portspeed_list=[4, 200000, 200000, 200000,␣
↪200000])
await module.cfp.config.set(portspeed_list=[8, 100000, 100000, 100000,␣
↪100000, 100000, 100000, 100000, 100000])

resp = await module.cfp.config.get()
resp.portspeed_list
```

Obtain

Obtain One

```
module = tester.modules.obtain(idx)
```

Obtain Multiple

```
module_list = tester.modules.obtain_multiple(*[idx_a, idx_b, idx_c])
```

Reservation

Reservation Action

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.

Corresponding CLI command: M_RESERVATION

```
# Reservation
await module.reservation.set(operation=enums.ReservedAction.RELEASE)
await module.reservation.set_release()
await module.reservation.set(operation=enums.ReservedAction.RELINQUISH)
await module.reservation.set_relinquish()
await module.reservation.set(operation=enums.ReservedAction.RESERVE)
await module.reservation.set_reserve()

resp = await module.reservation.get()
resp.operation
```

Reserved By

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

Corresponding CLI command: `M_RESERVEDBY`

```
# Reserved By
resp = await module.reserved_by.get()
resp.username
```

Time and Clock

Local Clock Adjust

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

Corresponding CLI command: `M_CLOCKPPB`

```
# Local Clock Adjust
await module.timing.clock_local_adjust.set(ppb=10)

resp = await module.timing.clock_local_adjust.get()
resp.ppb
```

Clock Sync Status

Get module's clock sync status.

Corresponding CLI command: `M_CLOCKSYNCSTATUS`

```
# Clock Sync Status
resp = await module.timing.clock_sync_status.get()
resp.m_clock_diff
resp.m_correction
resp.m_is_steady_state
resp.m_tune_is_increase
resp.m_tune_value
```

Clock Source

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.

Corresponding CLI command: `M_TIMESYNC`

```
# Clock Source
await module.timing.source.set(source=enums.TimingSource.CHASSIS)
await module.timing.source.set_chassis()
await module.timing.source.set(source=enums.TimingSource.EXTERNAL)
await module.timing.source.set_external()
await module.timing.source.set(source=enums.TimingSource.MODULE)
await module.timing.source.set_module()

resp = await module.timing.source.get()
resp.source
```

Clock PPM Sweep Configuration

Important: For Freya modules only

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

The sweep is independent of the M_CLOCKPPB parameter, i.e. the sweep uses the deviation set by M_CLOCKPPB as its zero point.

Corresponding CLI command: `M_CLOCKPPBSWEEP`

```
# Clock PPM Sweep Configuration
FREYA_MODULES = (modules.MFreya800G4S1P_a, modules.MFreya800G4S1P_b,␣
↪modules.MFreya800G4S1POSFP_a, modules.MFreya800G4S1POSFP_b)
if isinstance(module, FREYA_MODULES):
    await module.clock_sweep.config.set(mode=enums.PPMSweepMode.OFF,␣
↪ppb_step=10, step_delay=10, max_ppb=10, loops=1)
    await module.clock_sweep.config.set(mode=enums.PPMSweepMode.
↪TRIANGLE, ppb_step=10, step_delay=10, max_ppb=10, loops=1)

    resp = await module.clock_sweep.config.get()
    resp.mode
    resp.ppb_step
    resp.step_delay
    resp.max_ppb
    resp.loops
```

Clock PPM Sweep Status

Return the current status of the M_CLOCKPPBSWEEP command.

Corresponding CLI command: `M_CLOCKSWEEPSTATUS`

```
# Clock PPM Sweep Status
if isinstance(module, FREYA_MODULES):
    resp = await module.clock_sweep.status.get()
    resp.curr_step
    resp.curr_sweep
    resp.max_steps
```

Advanced Timing

TX Clock Filter Loop Bandwidth

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

Corresponding CLI command: `M_TXCLOCKFILTER_NEW`

```
# TX Clock Filter Loop Bandwidth
await module.advanced_timing.clock_tx.filter.set(filter_
↪bandwidth=enums.LoopBandwidth.BW103HZ)
await module.advanced_timing.clock_tx.filter.set_bw103hz()
await module.advanced_timing.clock_tx.filter.set(filter_
↪bandwidth=enums.LoopBandwidth.BW1683HZ)
await module.advanced_timing.clock_tx.filter.set_bw1683hz()
await module.advanced_timing.clock_tx.filter.set(filter_
```

(continues on next page)

(continued from previous page)

```
↪bandwidth=enums.LoopBandwidth.BW207HZ)
await module.advanced_timing.clock_tx.filter.set_bw207hz()
await module.advanced_timing.clock_tx.filter.set(filter_
↪bandwidth=enums.LoopBandwidth.BW416HZ)
await module.advanced_timing.clock_tx.filter.set_bw416hz()
await module.advanced_timing.clock_tx.filter.set(filter_
↪bandwidth=enums.LoopBandwidth.BW7019HZ)
await module.advanced_timing.clock_tx.filter.set_bw7019hz()

resp = await module.advanced_timing.clock_tx.filter.get()
resp.filter_bandwidth
```

TX Clock Source

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

Corresponding CLI command: `M_TXCLOCKSOURCE_NEW`

```
# TX Clock Source
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.MODULELOCALCLOCK)
await module.advanced_timing.clock_tx.source.set_modulelocalclock()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P0RXCLK)
await module.advanced_timing.clock_tx.source.set_p0rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P1RXCLK)
await module.advanced_timing.clock_tx.source.set_p1rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P2RXCLK)
await module.advanced_timing.clock_tx.source.set_p2rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P3RXCLK)
await module.advanced_timing.clock_tx.source.set_p3rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P4RXCLK)
await module.advanced_timing.clock_tx.source.set_p4rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P5RXCLK)
await module.advanced_timing.clock_tx.source.set_p5rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P6RXCLK)
await module.advanced_timing.clock_tx.source.set_p6rxclk()
await module.advanced_timing.clock_tx.source.set(tx_clock=enums.
↪TXClockSource.P7RXCLK)
```

(continues on next page)

(continued from previous page)

```
await module.advanced_timing.clock_tx.source.set_p7rxclk()

resp = await module.advanced_timing.clock_tx.source.get()
resp.tx_clock
```

TX Clock Status

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

Corresponding CLI command: `M_TXCLOCKSTATUS_NEW`

```
# TX Clock Status
resp = await module.advanced_timing.clock_tx.status.get()
resp.status
```

SMA Status

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

Corresponding CLI command: `M_SMASTATUS`

```
# SMA Status
resp = await module.advanced_timing.sma.status.get()
resp.status
```

SMA Input

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

Corresponding CLI command: `M_SMAINPUT`

```
# SMA Input
await module.advanced_timing.sma.input.set(sma_in=enums.
↪SMAInputFunction.NOT_USED)
await module.advanced_timing.sma.input.set_notused()
await module.advanced_timing.sma.input.set(sma_in=enums.
↪SMAInputFunction.TX10MHZ)
await module.advanced_timing.sma.input.set_tx10mhz()
await module.advanced_timing.sma.input.set(sma_in=enums.
↪SMAInputFunction.TX2MHZ)
await module.advanced_timing.sma.input.set_tx2mhz()
```

(continues on next page)

(continued from previous page)

```
resp = await module.advanced_timing.sma.input.get()
resp.sma_in
```

SMA Output

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

Corresponding CLI command: `M_SMAOUTPUT`

```
# SMA Output
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.DISABLED)
await module.advanced_timing.sma.output.set_disabled()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P0RXCLK)
await module.advanced_timing.sma.output.set_p0rxclk()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P0RXCLK2MHZ)
await module.advanced_timing.sma.output.set_p0rxclk2mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P0SOF)
await module.advanced_timing.sma.output.set_p0sof()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P1RXCLK)
await module.advanced_timing.sma.output.set_p1rxclk()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P1RXCLK2MHZ)
await module.advanced_timing.sma.output.set_p1rxclk2mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.P1SOF)
await module.advanced_timing.sma.output.set_p1sof()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.PASSTHROUGH)
await module.advanced_timing.sma.output.set_passthrough()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF10MHZ)
await module.advanced_timing.sma.output.set_ref10mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF125MHZ)
await module.advanced_timing.sma.output.set_ref125mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.REF156MHZ)
await module.advanced_timing.sma.output.set_ref156mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
```

(continues on next page)

(continued from previous page)

```
↪SMAOutputFunction.REF2MHZ)
await module.advanced_timing.sma.output.set_ref2mhz()
await module.advanced_timing.sma.output.set(sma_out=enums.
↪SMAOutputFunction.TS_PPS)
await module.advanced_timing.sma.output.set_ts_pps()

resp = await module.advanced_timing.sma.output.get()
resp.sma_out
```

L47

Note: Applicable to Vulcan module only.

Capture

Note: For Vulcan module only.

List PCAP Files

```
await module.capture.file_list.get()
```

List PCAP BSON Files

```
await module.capture.file_list_bson.get()
```

Delete PCAP Files

```
await module.capture.file_delete.set()
```

PCAP Parser Configuration

```
await module.capture.parse.parser_params.set()
await module.capture.parse.parser_params.get()
```

Start PCAP Parser

```
await module.capture.parse.start.set()
```

Stop PCAP Parser

```
await module.capture.parse.stop.set()
```

PCAP Parser State

```
await module.capture.parse.state.get()
```

Buffer Size

```
await module.capture.size.set_full()
await module.capture.size.set_small()
await module.capture.size.get()
```

License

Note: For Vulcan module only.

Clock Windback

```
await module.license.clock_windback.get()
```

Demo Information

```
await module.license.demo_info.get()
```

Online Mode

```
await module.license.online_mode.set_offline()
await module.license.online_mode.set_online()
await module.license.online_mode.get()
```

Update

```
await module.license.update.set()
```

Update Status

```
await module.license.update_status.get()
```

Management Information

```
await module.license.management_info.get()
```

List License BSON Files

```
await module.license.list_bson.get()
```

Packet Engine

Note: For Vulcan module only.

License Information

```
await module.packet_engine.license_info.get()
```

Mode

```
await module.packet_engine.mode.set_advanced()
await module.packet_engine.mode.set_simple()
await module.packet_engine.mode.get()
```

Reservation

```
await module.packet_engine.reserve.set()
await module.packet_engine.reserve.get()
```

Update

```
await module.license.update.set()
```

Update Status

```
await module.license.update_status.get()
```

Management Information

```
await module.license.management_info.get()
```

List License BSON Files

```
await module.license.list_bson.get()
```

PCAP Replay

Note: For Vulcan module only.

List Replay Files

```
await module.replay.file.list.get()
```

List Replay BSON Files

```
await module.replay.file.list_bson.get()
```

Delete Replay File

```
await module.replay.file.delete.set()
```

System

Note: For Vulcan module only.

Memory Information

```
await module.memory_info.get()
```

Identifier

```
await module.module_system.id.get()
```

Status

```
await module.module_system.status.get()
```

Time

```
await module.module_system.time.set()
await module.module_system.time.get()
```

Compatible Client Version

```
await module.compatible_client_version.get()
```

TLS Cipher

Note: For Vulcan module only.

```
await module.tls_cipher.get()
```

8.2.4 Port

Package `xoa_driver.ports` includes port APIs for Valkyrie, Vulcan, Chimera, ValkyrieVE, and VulcanVE.

Action

Reset

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

Corresponding CLI command: `P_RESET`

```
# Reset
await port.reset.set()
```

Flash

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.

Corresponding CLI command: P_FLASH

```
# Flash
await port.flash.set(on_off=enums.OnOff.ON)
await port.flash.set_on()
await port.flash.set(on_off=enums.OnOff.OFF)
await port.flash.set_off()

resp = await port.flash.get()
resp.on_off
```

Port Address

MAC Address

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.

Corresponding CLI command: P_MACADDRESS

```
# MAC Address
await port.net_config.mac_address.set(mac_address=Hex("000000000000"))

resp = await port.net_config.mac_address.get()
resp.mac_address
```

IPv4 Address

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.

Corresponding CLI command: P_IPADDRESS

```
# IPv4 Address
await port.net_config.ipv4.address.set(
    ipv4_address=ipaddress.IPv4Address("10.10.10.10"),
    subnet_mask=ipaddress.IPv4Address("255.255.255.0"),
    gateway=ipaddress.IPv4Address("10.10.1.1"),
    wild=ipaddress.IPv4Address("0.0.0.0"))
```

(continues on next page)

(continued from previous page)

```
resp = await port.net_config.ipv4.address.get()
resp.ipv4_address
resp.gateway
resp.subnet_mask
resp.wild
```

ARP Reply

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.

Corresponding CLI command: `P_ARPREPLY`

```
# ARP Reply
await port.net_config.ipv4.arp_reply.set(on_off=enums.OnOff.ON)
await port.net_config.ipv4.arp_reply.set(on_off=enums.OnOff.OFF)

resp = await port.net_config.ipv4.arp_reply.get()
resp.on_off
```

Ping Reply

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.

Corresponding CLI command: `P_PINGREPLY`

```
# Ping Reply
await port.net_config.ipv4.ping_reply.set(on_off=enums.OnOff.ON)
await port.net_config.ipv4.ping_reply.set(on_off=enums.OnOff.OFF)

resp = await port.net_config.ipv4.ping_reply.get()
resp.on_off
```

IPv6 Address

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.

Corresponding CLI command: `P_IPV6ADDRESS`

```
# IPv6 Address
await port.net_config.ipv6.address.set(
    ipv6_address=ipaddress.IPv6Address("fc00::0002"),
    gateway=ipaddress.IPv6Address("fc00::0001"),
    subnet_prefix=7,
    wildcard_prefix=0
)

resp = await port.net_config.ipv6.address.get()
resp.ipv6_address
resp.gateway
resp.subnet_prefix
resp.wildcard_prefix
```

NDP Reply

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.

Corresponding CLI command: `P_ARPV6REPLY`

```
# NDP Reply
await port.net_config.ipv6.arp_reply.set(on_off=enums.OnOff.ON)
await port.net_config.ipv6.arp_reply.set(on_off=enums.OnOff.OFF)

resp = await port.net_config.ipv6.arp_reply.get()
resp.on_off
```

IPv6 Ping Reply

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.

Corresponding CLI command: `P_PINGV6REPLY`

```
# IPv6 Ping Reply
await port.net_config.ipv6.ping_reply.set(on_off=enums.OnOff.ON)
await port.net_config.ipv6.ping_reply.set(on_off=enums.OnOff.OFF)

resp = await port.net_config.ipv6.ping_reply.get()
resp.on_off
```

ARP Table

Port ARP table used to reply to incoming ARP requests.

Corresponding CLI command: `P_ARPRXTABLE`

See also:

Detailed script example can be found at ip_streams_arp_table

```
# ARP Table
await port.arp_rx_table.set(chunks=[])

resp = await port.arp_rx_table.get()
resp.chunks
```

NDP Table

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

Corresponding CLI command: `P_NDPRXTABLE`

See also:

Detailed script example can be found at ip_streams_arp_table

```
# NDP Table
await port.ndp_rx_table.set(chunks=[])

resp = await port.ndp_rx_table.get()
resp.chunks
```

Capabilities

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

Corresponding CLI command: P_CAPABILITIES

```
await port.capabilities.get()
```

Capture

Trigger Criteria

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).

Corresponding CLI command: PC_TRIGGER

See also:

Detailed script example can be found in here

```
# Capture Trigger Criteria,
await port.capturer.trigger.set(start_criteria=enums.StartTrigger.ON,␣
↪start_criteria_filter=0, stop_criteria=enums.StopTrigger.FULL, stop_
↪criteria_filter=0)
await port.capturer.trigger.set(start_criteria=enums.StartTrigger.ON,␣
↪start_criteria_filter=0, stop_criteria=enums.StopTrigger.USERSTOP,␣
↪stop_criteria_filter=0)
await port.capturer.trigger.set(start_criteria=enums.StartTrigger.
↪FCSERR, start_criteria_filter=0, stop_criteria=enums.StopTrigger.
↪FCSERR, stop_criteria_filter=0)
await port.capturer.trigger.set(start_criteria=enums.StartTrigger.
↪PLDERR, start_criteria_filter=0, stop_criteria=enums.StopTrigger.
↪PLDERR, stop_criteria_filter=0)
await port.capturer.trigger.set(start_criteria=enums.StartTrigger.
↪FILTER, start_criteria_filter=0, stop_criteria=enums.StopTrigger.
↪FILTER, stop_criteria_filter=0)

resp = await port.capturer.trigger.get()
resp.start_criteria
resp.start_criteria_filter
resp.stop_criteria
resp.stop_criteria_filter
```

Frame to Keep

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.

See also:

Detailed script example can be found in here

Corresponding CLI command: `PC_KEEP`

```
# Capture - Frame to Keep,
await port.capturer.keep.set(kind=enums.PacketType.ALL, index=0, byte_
↪count=0)
await port.capturer.keep.set_all()
await port.capturer.keep.set(kind=enums.PacketType.FCSERR, index=0,␣
↪byte_count=0)
await port.capturer.keep.set_fcserr()
await port.capturer.keep.set(kind=enums.PacketType.FILTER, index=0,␣
↪byte_count=0)
await port.capturer.keep.set_filter()
await port.capturer.keep.set(kind=enums.PacketType.NOTPLD, index=0,␣
↪byte_count=0)
await port.capturer.keep.set_notpld()
await port.capturer.keep.set(kind=enums.PacketType.PLDERR, index=0,␣
↪byte_count=0)
await port.capturer.keep.set_plderr()
await port.capturer.keep.set(kind=enums.PacketType.TPLD, index=0, byte_
↪count=0)
await port.capturer.keep.set_tpld()

resp = await port.capturer.keep.get()
resp.kind
resp.index
resp.byte_count
```

State

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.

Corresponding CLI command: `P_CAPTURE`

```
# Capture - State
await port.capturer.state.set(on_off=enums.StartOrStop.START)
await port.capturer.state.set_start()
await port.capturer.state.set(on_off=enums.StartOrStop.STOP)
```

(continues on next page)

(continued from previous page)

```
await port.capturer.state.set_stop()

resp = await port.capturer.state.get()
resp.on_off
```

Statistics

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

Corresponding CLI command: PC_STATS

```
# Capture - Statistics
resp = await port.capturer.stats.get()
resp.start_time
resp.status
```

Read Captured Packets

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.

Corresponding CLI command: PC_PACKET

```
# Read Captured Packets
pkts = await port.capturer.obtain_captured()
for i in range(len(pkts)):
    resp = await pkts[i].packet.get()
    print(f"Packet content # {i}: {resp.hex_data}")
```

Control

Inter-frame Gap

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

Corresponding CLI command: P_INTERFRAMEGAP

```
# Inter-frame Gap
await port.interframe_gap.set(min_byte_count=20)

resp = await port.interframe_gap.get()
resp.min_byte_count
```

PAUSE Frames

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

Corresponding CLI command: `P_PAUSE`

```
# PAUSE Frames
await port.pause.set(on_off=enums.OnOff.ON)
await port.pause.set_on()
await port.pause.set(on_off=enums.OnOff.OFF)
await port.pause.set_off()

resp = await port.pause.get()
resp.on_off
```

Auto-Train

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.

Corresponding CLI command: `P_AUTOTRAIN`

```
# Auto-Train
await port.autotrain.set(interval=1)

resp = await port.autotrain.get()
resp.interval
```

Gap Monitor

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.

Corresponding CLI command: `P_GAPMONITOR`

```
# Gap Monitor
await port.gap_monitor.set(start=100, stop=10)

resp = await port.gap_monitor.get()
```

(continues on next page)

(continued from previous page)

```
resp.start
resp.stop
```

Priority Flow Control

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

Corresponding CLI command: `P_PFCENABLE`

```
# Priority Flow Control
await port.pfc_enable.set(
    cos_0=enums.OnOff.ON,
    cos_1=enums.OnOff.OFF,
    cos_2=enums.OnOff.ON,
    cos_3=enums.OnOff.OFF,
    cos_4=enums.OnOff.ON,
    cos_5=enums.OnOff.OFF,
    cos_6=enums.OnOff.ON,
    cos_7=enums.OnOff.OFF,
    )

resp = await port.pfc_enable.get()
resp.cos_0
resp.cos_1
resp.cos_2
resp.cos_3
resp.cos_4
resp.cos_5
resp.cos_6
resp.cos_7
```

Loopback

The loopback mode for a port. Ports can be configured to perform two different kinds of loopback: (1) External RX-to-TX loopback, where the received packets are re-transmitted immediately. The packets are still processed by the receive logic, and can be captured and analyzed. (2) Internal TX-to-RX loopback, where the transmitted packets are received directly by the port itself. This is mainly useful for testing the generated traffic patterns before actual use.

Corresponding CLI command: `P_LOOPBACK`

```
# Loopback
await port.loop_back.set(mode=enums.LoopbackMode.L1RX2TX)
await port.loop_back.set_l1rx2tx()
```

(continues on next page)

(continued from previous page)

```
await port.loop_back.set(mode=enums.LoopbackMode.L2RX2TX)
await port.loop_back.set_l2rx2tx()
await port.loop_back.set(mode=enums.LoopbackMode.L3RX2TX)
await port.loop_back.set_l3rx2tx()
await port.loop_back.set(mode=enums.LoopbackMode.NONE)
await port.loop_back.set_none()
await port.loop_back.set(mode=enums.LoopbackMode.PORT2PORT)
await port.loop_back.set_port2port()
await port.loop_back.set(mode=enums.LoopbackMode.TXOFF2RX)
await port.loop_back.set_txoff2rx()
await port.loop_back.set(mode=enums.LoopbackMode.TXON2RX)
await port.loop_back.set_txon2rx()

resp = await port.loop_back.get()
resp.mode
```

BRR Mode

Selects the Master/Slave setting of 100 Mbit/s, 1000 Mbit/s BroadR-Reach copper interfaces.

Corresponding CLI command: `P_BRRMODE`

```
# BRR Mode
await port.brr_mode.set(mode=enums.BRRMode.MASTER)
await port.brr_mode.set_master()
await port.brr_mode.set(mode=enums.BRRMode.SLAVE)
await port.brr_mode.set_slave()

resp = await port.brr_mode.get()
resp.mode
```

MDI/MDIX Mode

Selects the MDI/MDIX behavior of copper interfaces.

Corresponding CLI command: `P_MDIXMODE`

```
# MDI/MDIX Mode
await port.mdix_mode.set(mode=enums.MDIXMode.AUTO)
await port.mdix_mode.set_auto()
await port.mdix_mode.set(mode=enums.MDIXMode.MDI)
await port.mdix_mode.set_mdi()
await port.mdix_mode.set(mode=enums.MDIXMode.MDIX)
await port.mdix_mode.set_mdix()
```

(continues on next page)

(continued from previous page)

```
resp = await port.mdix_mode.get()
resp.mode
```

Energy Efficiency Ethernet

Capabilities

Read EEE capabilities of the port (variable size, one for each supported speed, returns 0s if no EEE).

Corresponding CLI command: P_LPSUPPORT

```
# EEE- Capabilities
resp = await port.eee.capabilities.get()
resp.eee_capabilities
```

Partner Capabilities

Displays the EEE capabilities advertised during auto-negotiation by the far side (link partner).

Corresponding CLI command: P_LPPARTNERAUTONEG

```
# EEE - Partner Capabilities
resp = await port.eee.partner_capabilities.get()
resp.cap_1000base_t
resp.cap_100base_kx
resp.cap_10gbase_kr
resp.cap_10gbase_kx4
resp.cap_10gbase_t
```

Control

Enables/disables Energy Efficient Ethernet (EEE) on the port.

Corresponding CLI command: P_LPENABLE

```
# EEE - Control
await port.eee.enable.set(on_off=enums.OnOff.OFF)
await port.eee.enable.set_off()
await port.eee.enable.set(on_off=enums.OnOff.ON)
await port.eee.enable.set_on()

resp = await port.eee.enable.get()
resp.on_off
```

Low Power TX Mode

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.

Corresponding CLI command: `P_LPTXMODE`

```
# EEE - Low Power TX Mode
await port.eee.mode.set(on_off=enums.OnOff.ON)
await port.eee.mode.set_off()
await port.eee.mode.set(on_off=enums.OnOff.OFF)
await port.eee.mode.set_on()

resp = await port.eee.mode.get()
resp.on_off
```

RX Power

Obtain the RX power recorded during training for the four channels.

Corresponding CLI command: `P_LPRXPOWER`

```
# EEE - RX Power
resp = await port.eee.rx_power.get()
resp.channel_a
resp.channel_b
resp.channel_c
resp.channel_d
```

SNR Margin

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.

Corresponding CLI command: `P_LPSNRMARGIN`

```
# EEE - SNR Margin
resp = await port.eee.snr_margin.get()
resp.channel_a
resp.channel_b
resp.channel_c
resp.channel_d
```

Status

Displays the Energy Efficient Ethernet (EEE) status as reported by the PHY.

Corresponding CLI command: P_LPSTATUS

```
# EEE - Status
resp = await port.eee.status.get()
resp.link_up
resp.rxc
resp.rxh
resp.txc
resp.txh
```

Fault

Signaling

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.

Corresponding CLI command: P_FAULTSIGNALING

```
# Fault - Signaling
await port.fault.signaling.set(fault_signaling=enums.FaultSignaling.
↪DISABLED)
await port.fault.signaling.set_disabled()
await port.fault.signaling.set(fault_signaling=enums.FaultSignaling.
↪FORCE_LOCAL)
await port.fault.signaling.set_force_local()
await port.fault.signaling.set(fault_signaling=enums.FaultSignaling.
↪FORCE_REMOTE)
await port.fault.signaling.set_force_remote()
await port.fault.signaling.set(fault_signaling=enums.FaultSignaling.
↪NORMAL)
await port.fault.signaling.set_normal()

resp = await port.fault.signaling.get()
resp.fault_signaling
```

Status

Shows if a local or remote fault is currently being detected by the Reconciliation Sub-layer of the port.

Corresponding CLI command: `P_FAULTSTATUS`

```
# Fault - Status
resp = await port.fault.status.get()
resp.local_fault_status
resp.remote_fault_status
```

Identification

Interface

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

Corresponding CLI command: `P_INTERFACE`

```
# Interface
resp = await port.interface.get()
resp.interface
```

Description

The description of a port.

Corresponding CLI command: `P_COMMENT`

```
# Description
await port.comment.set(comment="description")

resp = await port.comment.get()
resp.comment
```

Optical Signal Level

Get the received signal level for optical ports.

Corresponding CLI command: `P_STATUS`

```
# Status
resp = await port.status.get()
resp.optical_power
```

Latency

Mode

Latency is measured by inserting a time-stamp in each packet when it is transmitted, and relating it to the time when the packet is received. There are four separate modes for calculating the latency:

1. Last-bit-out to last-bit-in, which measures basic bit-transit time, independent of packet length.
2. First-bit-out to last-bit-in, which adds the time taken to transmit the packet itself.
3. 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.
4. 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.

Corresponding CLI command: `P_LATENCYMODE`

```
# Latency Mode
await port.latency_config.mode.set(mode=enums.LatencyMode.FIRST2FIRST)
await port.latency_config.mode.set_first2first()
await port.latency_config.mode.set(mode=enums.LatencyMode.FIRST2LAST)
await port.latency_config.mode.set_first2last()
await port.latency_config.mode.set(mode=enums.LatencyMode.LAST2FIRST)
await port.latency_config.mode.set_last2first()
await port.latency_config.mode.set(mode=enums.LatencyMode.LAST2LAST)
await port.latency_config.mode.set_last2last()

resp = await port.latency_config.mode.get()
resp.mode
```

Offset

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.

Corresponding CLI command: `P_LATENCYOFFSET`

```
# Latency Offset
await port.latency_config.offset.set(offset=5)
```

(continues on next page)

(continued from previous page)

```
resp = await port.latency_config.offset.get()
resp.offset
```

Link Flap

Control

Enable / disable port 'link flap'.

Corresponding CLI command: `PP_LINKFLAP_ENABLE`

```
# Link Flap - Control
await port.pcs_pma.link_flap.enable.set(on_off=enums.OnOff.ON)
await port.pcs_pma.link_flap.enable.set_on()
await port.pcs_pma.link_flap.enable.set(on_off=enums.OnOff.OFF)
await port.pcs_pma.link_flap.enable.set_off()

resp = await port.pcs_pma.link_flap.enable.get()
resp.on_off
```

Configuration

Set port 'link flap' parameters. Notice: Period must be larger than duration.

Corresponding CLI command: `PP_LINKFLAP_PARAMS`

```
# Link Flap - Configuration
await port.pcs_pma.link_flap.params.set(duration=10, period=20,␣
↪repetition=0)

resp = await port.pcs_pma.link_flap.params.get()
resp.duration
resp.period
resp.repetition
```

Multicast

Mode

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.

Corresponding CLI command: `P_MULTICAST`

```
# Multicast Mode
await port.multicast.mode.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastOperation.JOIN,
    second_count=10)
await port.multicast.mode.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastOperation.JOIN,
    second_count=10)
await port.multicast.mode.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastOperation.LEAVE,
    second_count=10)
await port.multicast.mode.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastOperation.OFF,
    second_count=10)
await port.multicast.mode.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastOperation.ON,
    second_count=10)

resp = await port.multicast.mode.get()
resp.ipv4_multicast_addresses
resp.operation
resp.second_count
```

Extended Mode

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.

Corresponding CLI command: `P_MULTICASTEXT`

```
# Multicast Extended Mode
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.EXCLUDE,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV3
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.INCLUDE,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV3
```

(continues on next page)

(continued from previous page)

```
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.JOIN,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.LEAVE,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.LEAVE_TO_ALL,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.GENERAL_QUERY,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.GROUP_QUERY,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.ON,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)
await port.multicast.mode_extended.set(
    ipv4_multicast_addresses=[],
    operation=enums.MulticastExtOperation.OFF,
    second_count=10,
    igmp_version=enums.IGMPVersion.IGMPV2
)

resp = await port.multicast.mode_extended.get()
resp.ipv4_multicast_addresses
```

(continues on next page)

(continued from previous page)

```
resp.operation
resp.second_count
resp.igmp_version
```

Source List

Multicast source list of the port. Only valid if the IGMP protocol version is IGMPv3 set by P_MULTICASTEXT.

Corresponding CLI command: `P_MCSRCLIST`

```
# Multicast Source List
await port.multicast.source_list.set(ipv4_addresses=[])

resp = await port.multicast.source_list.get()
resp.ipv4_addresses
```

Header

Allows addition of a VLAN tag to IGMPv2 and IGPMv3 packets.

Corresponding CLI command: `P_MULTICASTHDR`

```
# Multicast Header
await port.multicast.header.set(header_count=1, header_format=enums.
↪MulticastHeaderFormat.VLAN, tag=10, pcp=0, dei=0)
await port.multicast.header.set(header_count=0, header_format=enums.
↪MulticastHeaderFormat.NOHDR, tag=10, pcp=0, dei=0)

resp = await port.multicast.header.get()
resp.header_count
resp.header_format
resp.tag
resp.pcp
resp.dei
```

Obtain

Obtain One

```
port = module.ports.obtain(idx)
```

Obtain Multiple

```
port_list = module.ports.obtain_multiple(*[idx_a, idx_b, idx_c ...])
```

Payload

Random Seed

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.

Corresponding CLI command: `P_RANDOMSEED`

```
# Random Seed
await port.random_seed.set(seed=1)

resp = await port.random_seed.get()
resp.seed
```

Checksum Offset

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).

Corresponding CLI command: `P_CHECKSUM`

```
# Checksum Offset
await port.checksum.set(offset=14)

resp = await port.checksum.get()
resp.offset
```

Maximum Header Length

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.

Corresponding CLI command: `P_MAXHEADERLENGTH`

```
# Maximum Header Length
await port.max_header_length.set(max_header_length=56)

resp = await port.max_header_length.get()
resp.max_header_length
```

MIX Weights

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.

Corresponding CLI command: `P_MIXWEIGHTS`

```
# MIX Weights
await port.mix.weights.set(
    weight_56_bytes:=0,
    weight_60_bytes:=0,
    weight_64_bytes:=70,
    weight_70_bytes:=15,
    weight_78_bytes:=15,
    weight_92_bytes:=0,
    weight_256_bytes:=0,
    weight_496_bytes:=0,
    weight_512_bytes:=0,
    weight_570_bytes:=0,
    weight_576_bytes:=0,
    weight_594_bytes:=0,
    weight_1438_bytes:=0,
    weight_1518_bytes:=0,
    weight_9216_bytes:=0,
    weight_16360_bytes:=0)

resp = await port.mix.weights.get()
resp.weight_56_bytes
```

(continues on next page)

(continued from previous page)

```
resp.weight_60_bytes
resp.weight_64_bytes
resp.weight_70_bytes
resp.weight_78_bytes
resp.weight_92_bytes
resp.weight_256_bytes
resp.weight_496_bytes
resp.weight_512_bytes
resp.weight_570_bytes
resp.weight_576_bytes
resp.weight_594_bytes
resp.weight_1438_bytes
resp.weight_1518_bytes
resp.weight_9216_bytes
resp.weight_16360_bytes
```

MIX Lengths

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).

Corresponding CLI command: `P_MIXLENGTH`

```
# MIX Lengths
await port.mix.lengths[0].set(frame_size=56)
await port.mix.lengths[1].set(frame_size=60)
await port.mix.lengths[14].set(frame_size=9216)
await port.mix.lengths[15].set(frame_size=16360)

resp = await port.mix.lengths[0].get()
resp.frame_size
resp = await port.mix.lengths[1].get()
resp.frame_size
resp = await port.mix.lengths[14].get()
resp.frame_size
resp = await port.mix.lengths[15].get()
resp.frame_size
```

Payload Mode

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.

Corresponding CLI command: P_PAYLOADMODE

```
# Payload Mode
await port.payload_mode.set(mode=enums.PayloadMode.NORMAL)
await port.payload_mode.set_normal()
await port.payload_mode.set(mode=enums.PayloadMode.EXTPL)
await port.payload_mode.set_extpl()
await port.payload_mode.set(mode=enums.PayloadMode.CDF)
await port.payload_mode.set_cdf()

resp = await port.payload_mode.get()
resp.mode
```

Preamble

RX Preamble Insert

Insert preambles to the incoming frames.

Corresponding CLI command: P_RXPREAMBLE_INSERT

```
# RX Preamble Insert
await port.preamble.rx_insert.set(on_off=enums.OnOff.ON)
await port.preamble.rx_insert.set(on_off=enums.OnOff.OFF)

resp = await port.preamble.rx_insert.get()
resp.on_off
```

TX Preamble Removal

Remove preamble from outgoing frames.

Corresponding CLI command: P_TXPREAMBLE_REMOVE

```
# TX Preamble Removal
await port.preamble.tx_remove.set(on_off=enums.OnOff.ON)
await port.preamble.tx_remove.set(on_off=enums.OnOff.OFF)
```

(continues on next page)

(continued from previous page)

```
resp = await port.preamble.tx_remove.get()
resp.on_off
```

Reservation

Action

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.

Corresponding CLI command: `P_RESERVATION`

```
# Reservation
await port.reservation.set(operation=enums.ReservedAction.RELEASE)
await port.reservation.set_release()
await port.reservation.set(operation=enums.ReservedAction.RELINQUISH)
await port.reservation.set_relinquish()
await port.reservation.set(operation=enums.ReservedAction.RESERVE)
await port.reservation.set_reserve()

resp = await port.reservation.get()
resp.status
```

Reserved By

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.

Corresponding CLI command: `P_RESERVEDBY`

```
# Reserved By
resp = await port.reserved_by.get()
resp.username
```

Runt

RX Length

Enable RX runt length detection to flag if packets are seen with length not being I bytes.

Corresponding CLI command: P_RXRUNTLENGTH

```
# Runt - RX Length
await port.runt.rx_length.set(runt_length=40)

resp = await port.runt.rx_length.get()
resp.runt_length
```

TX Length

Enable TX runt feature to cut all packets to a number of bytes.

Corresponding CLI command: P_TXRUNTLENGTH

```
# Runt - TX Length
await port.runt.tx_length.set(runt_length=40)

resp = await port.runt.tx_length.get()
resp.runt_length
```

Length Error

Sticky clear on read: Have packets with wrong runt length been detected since last read?

Corresponding CLI command: P_RXRUNTLEN_ERRS

```
# Runt - Length Error
resp = await port.runt.has_length_errors.get()
resp.status
```

Speed

Mode Selection

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_CFPCONFIGEXT` command when speed is selected at the module level.

Corresponding CLI command: `P_SPEEDSELECTION`

```
# Speed Mode Selection
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.AUTO)
await port.speed.mode.selection.set_auto()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10M)
await port.speed.mode.selection.set_f10m()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10M100M)
await port.speed.mode.selection.set_f10m100m()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10MHDX)
await port.speed.mode.selection.set_f10mhdx()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100M)
await port.speed.mode.selection.set_f100m()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100M1G)
await port.speed.mode.selection.set_f100m1g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F100M1G10G)
await port.speed.mode.selection.set_f100m1g10g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.
↪F100M1G2500M)
await port.speed.mode.selection.set_f100m1g2500m()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100MHDX)
await port.speed.mode.selection.set_f100mhdx()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F1G)
await port.speed.mode.selection.set_f1g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F2500M)
await port.speed.mode.selection.set_f2500m()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F5G)
await port.speed.mode.selection.set_f5g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F10G)
await port.speed.mode.selection.set_f10g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F40G)
await port.speed.mode.selection.set_f40g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F100G)
await port.speed.mode.selection.set_f100g()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.UNKNOWN)
await port.speed.mode.selection.set_unknown()
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F200G)
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F400G)
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F800G)
await port.speed.mode.selection.set(mode=enums.PortSpeedMode.F1600G)

resp = await port.speed.mode.selection.get()
resp.mode
```

Supported Modes

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.

Corresponding CLI command: `P_SPEEDS_SUPPORTED`

```
# Supported Speed Modes
resp = await port.speed.mode.supported.get()
resp.auto
resp.f10M
resp.f100M
resp.f1G
resp.f10G
resp.f40G
resp.f100G
resp.f10MHDX
resp.f100MHDX
resp.f10M100M
resp.f100M1G
resp.f100M1G10G
resp.f2500M
resp.f5G
resp.f100M1G2500M
resp.f25G
resp.f50G
resp.f200G
resp.f400G
resp.f800G
resp.f1600G
```

Current Speed

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

Corresponding CLI command: `P_SPEED`

```
# Current Speed
resp = await port.speed.current.get()
resp.port_speed
```

Speed Reduction

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.

Corresponding CLI command: `P_SPEEDREDUCTION`

```
# Speed Reduction
await port.speed.reduction.set(ppm=100)

resp = await port.speed.reduction.get()
resp.ppm
```

Status

Sync Status

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

Corresponding CLI command: `P_RECEIVESYNC`

```
# Sync Status
resp = await port.sync_status.get()
resp.sync_status == enums.SyncStatus.IN_SYNC
resp.sync_status == enums.SyncStatus.NO_SYNC
```

Transceiver

Status

Get various tcvr status information. RX loss status of the individual RX optical lanes (only 4 lanes are supported currently).

Corresponding CLI command: `P_TCVRSTATUS`

```
# Transceiver Status
resp = await port.tcvr_status.get()
resp.rx_loss_lane_0
resp.rx_loss_lane_1
resp.rx_loss_lane_2
resp.rx_loss_lane_3
```

Read & Write

Provides read and write access to the register interface supported by the port transceiver. It is possible to both read and write register values.

Corresponding CLI command: PX_RW

```
# Transceiver Read & Write
await port.transceiver.access_rw(page_address=0, register_address=0).
↪set(value=Hex("FF"))

resp = await port.transceiver.access_rw(page_address=0, register_
↪address=0).get()
resp.value
```

Sequential Read & Write

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_xindex>: the transceiver page address, integer, 0-255. <_register_xaddress>: the address within the page, integer, 0-255.

Corresponding CLI command: PX_RW_SEQ

```
# Transceiver Sequential Read & Write
await port.transceiver.access_rw_seq(page_address=0, register_
↪address=0, byte_count=4).set(value=Hex("00FF00FF"))

resp = await port.transceiver.access_rw_seq(page_address=0, register_
↪address=0, byte_count=4).get()
resp.value
```

MII

Provides access to the register interface supported by the media-independent interface (MII) transceiver. It is possible to both read and write register values.

Corresponding CLI command: PX_MII

```
# Transceiver MII
await port.transceiver.access_mii(register_address=0).set(value=Hex("00
↪"))
```

(continues on next page)

(continued from previous page)

```
resp = await port.transceiver.access_mii(register_address=0).get()
resp.value
```

Temperature

Transceiver temperature in degrees Celsius.

Corresponding CLI command: `PX_TEMPERATURE`

```
# Transceiver Temperature
resp = await port.transceiver.access_temperature().get()
resp.integral_part
resp.fractional_part
```

RX Laser Power

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

Corresponding CLI command: `PP_RXLASERPOWER`

```
# Transceiver RX Laser Power
resp = await port.pcs_pma.transceiver.rx_laser_power.get()
resp.nanowatts
```

TX Laser Power

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 kind of CFP transceiver used. The list is empty if the CFP transceiver does not support optical power read-out.

Corresponding CLI command: `PP_TXLASERPOWER`

```
# Transceiver TX Laser Power
resp = await port.pcs_pma.transceiver.tx_laser_power.get()
resp.nanowatts
```

Traffic Control

Rate Percent

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.

Corresponding CLI command: `P_RATEFRACTION`

```
# Traffic Control - Rate Percent
await port.rate.fraction.set(port_rate_ppm=1_000_000)

resp = await port.rate.fraction.get()
resp.port_rate_ppm
```

Rate L2 Bits Per Second

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.

Corresponding CLI command: `P_RATEL2BPS`

```
# Traffic Control - Rate L2 Bits Per Second
await port.rate.l2_bps.set(port_rate_bps=1_000_000)

resp = await port.rate.l2_bps.get()
resp.port_rate_bps
```

Rate Frames Per Second

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.

Corresponding CLI command: `P_RATEPPS`

```
# Traffic Control - Rate Frames Per Second
await port.rate.pps.set(port_rate_pps=10_000)

resp = await port.rate.pps.get()
resp.port_rate_pps
```

Start and Stop

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: 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.

Corresponding CLI command: `P_TRAFFIC`

```
# Traffic Control - Start and Stop
await port.traffic.state.set(on_off=enums.StartOrStop.START)
await port.traffic.state.set_start()
await port.traffic.state.set(on_off=enums.StartOrStop.STOP)
await port.traffic.state.set_stop()

resp = await port.traffic.state.get()
resp.on_off
```

Traffic Error

Obtain the traffic error which has occurred in the last `*_TRAFFIC` or `C_TRAFFICSYNC` command.

Corresponding CLI command: `P_TRAFFICERR`

```
# Traffic Control - Traffic Error
resp = await port.traffic.error.get()
resp.error
```

Single Frame TX

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.

Corresponding CLI command: `P_XMITONE`

```
# Traffic Control - Single Frame TX
await port.tx_single_pkt.send.set(hex_data=Hex(
↪"000000000001020304050608000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))
```

Single Frame Time

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.

Corresponding CLI command: `P_XMITONETIME`

```
# Traffic Control - Single Frame Time
resp = await port.tx_single_pkt.time.get()
resp.nanoseconds
```

TX Profile

TPLD Mode

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. When the TPLDMODE is changed, it will affect ALL streams on the port. 1) Only 6 byte long. 2) 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. 3) 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. Note: Currently not available on M6SFP, M2SFPT, M6RJ45+/M2RJ45+, M2CFP40, M1CFP100, M2SFP+4SFP

Corresponding CLI command: `P_TPLDMODE`

```
# TPLD Mode
await port.tpld_mode.set(mode=enums.TPLDMode.NORMAL)
await port.tpld_mode.set_normal()
await port.tpld_mode.set(mode=enums.TPLDMode.MICRO)
await port.tpld_mode.set_micro()

resp = await port.tpld_mode.get()
resp.mode
```

TX Mode

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.

Corresponding CLI command: `P_TXMODE`

```
# TX Mode
await port.tx_config.mode.set(mode=enums.TXMode.NORMAL)
await port.tx_config.mode.set_normal()
await port.tx_config.mode.set(mode=enums.TXMode.BURST)
await port.tx_config.mode.set_burst()
await port.tx_config.mode.set(mode=enums.TXMode.SEQUENTIAL)
await port.tx_config.mode.set_sequential()
await port.tx_config.mode.set(mode=enums.TXMode.STRICTUNIFORM)
await port.tx_config.mode.set_strictuniform()

resp = await port.tx_config.mode.get()
resp.mode
```

Burst Period

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.

Note: Only used when Port TX Mode is “BURST”.

Corresponding CLI command: `P_TXBURSTPERIOD`

```
# Burst Period
await port.tx_config.burst_period.set(burst_period=100)
```

(continues on next page)

(continued from previous page)

```
resp = await port.tx_config.burst_period.get()
resp.burst_period
```

TX Delay

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 take effect.

Corresponding CLI command: `P_TXDELAY`

```
# TX Delay
await port.tx_config.delay.set(delay_val=100)

resp = await port.tx_config.delay.get()
resp.delay_val
```

TX Enable

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

Corresponding CLI command: `P_TXENABLE`

```
# TX Enable
await port.tx_config.enable.set(on_off=enums.OnOff.ON)
await port.tx_config.enable.set(on_off=enums.OnOff.OFF)

resp = await port.tx_config.enable.get()
resp.on_off
```

Packet Limit

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.

Corresponding CLI command: `P_TXPACKETLIMIT`

```
# Packet Limit
await port.tx_config.packet_limit.set(packet_count_limit=1_000_000)

resp = await port.tx_config.packet_limit.get()
resp.packet_count_limit
```

Time Limit

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.

Corresponding CLI command: `P_TXTIMELIMIT`

```
# Time Limit
await port.tx_config.time_limit.set(microseconds=1_000_000)

resp = await port.tx_config.time_limit.get()
resp.microseconds
```

TX Time Elapsed

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

Corresponding CLI command: `P_TXTIME`

```
# TX Time Elapsed
resp = await port.tx_config.time.get()
resp.microseconds
```

Prepare TX

Prepare port for transmission.

Corresponding CLI command: `P_TXPREPARE`

```
# Prepare TX
await port.tx_config.prepare.set()
```

Dynamic TX Rate

Controls if a port with speed higher than 10G supports dynamic changes when the traffic is running.

Note: This command is only supported by ports with speed higher than 10G.

Corresponding CLI command: `P_DYNAMIC`

```
# Dynamic Traffic Rate
await port.dynamic.set(on_off=enums.OnOff.OFF)
await port.dynamic.set_off()
await port.dynamic.set(on_off=enums.OnOff.ON)
await port.dynamic.set_on()

resp = await port.dynamic.get()
resp.on_off
```

Unavailable Time

Mode

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.

Corresponding CLI command: `P_UAT_MODE`

```
await port.uat.mode.set(mode=enums.OnOff.ON, delay=500)
await port.uat.mode.set(mode=enums.OnOff.OFF, delay=500)

resp = await port.uat.mode.get()
resp.mode
resp.delay
```

Frame Loss Ratio

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

Corresponding CLI command: `P_UAT_FLR`

```
resp = await port.uat.frame_loss_ratio.get()
resp.frame_loss_ratio
```

Histogram

Histogram APIs for Valkyrie.

Configuration

Enable

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.

Corresponding CLI command: `PD_ENABLE`

```
await dataset.enable.set(on_off=enums.OnOff.ON)
await dataset.enable.set_on()
await dataset.enable.set(on_off=enums.OnOff.OFF)
await dataset.enable.set_off()

resp = await dataset.enable.get()
resp.on_off
```

Data Source

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

Corresponding CLI command: `PD_SOURCE`

```
await dataset.source.set(
    source_type=enums.SourceType.TX_IFG,
    which_packets=enums.PacketDetailSelection.ALL,
    identity=0
)
await dataset.source.set(
    source_type=enums.SourceType.TX_LEN,
    which_packets=enums.PacketDetailSelection.ALL,
    identity=0
)
await dataset.source.set(
    source_type=enums.SourceType.RX_IFG,
    which_packets=enums.PacketDetailSelection.ALL,
    identity=0
)
await dataset.source.set(
    source_type=enums.SourceType.RX_LEN,
    which_packets=enums.PacketDetailSelection.ALL,
```

(continues on next page)

(continued from previous page)

```
    identity=0
)
await dataset.source.set(
    source_type=enums.SourceType.RX_LATENCY,
    which_packets=enums.PacketDetailSelection.ALL,
    identity=0
)
await dataset.source.set(
    source_type=enums.SourceType.RX_JITTER,
    which_packets=enums.PacketDetailSelection.ALL,
    identity=0
)

resp = await dataset.source.get()
resp.source_type
resp.which_packets
resp.identity
```

Data Range

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.

Corresponding CLI command: `PD_RANGE`

```
await dataset.range.set(
    start=1, #first value going into the second bucket
    step=1, # 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).
    bucket_count=10 # the total number of buckets
)

resp = await dataset.range.get()
resp.start
resp.step
resp.bucket_count
```

Data Samples

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.

Corresponding CLI command: `PD_SAMPLES`

```
resp = await dataset.samples.get()
resp.packet_counts
```

Remove

Delete an existing histogram definition.

Corresponding CLI command: `PD_DELETE`

```
# Remove a histogram on the port with an explicit histogram index by␣
↪the index manager of the port.
await port.datasets.remove(position_idx=0)
```

Create, Obtain, Remove

Create and Obtain

Create a histogram on the port, and obtain the histogram object. The histogram index is automatically assigned by the port.

Corresponding CLI command: `PD_CREATE`

```
dataset = await port.datasets.create()
```

Obtain One

Obtain an existing histogram on the port with an explicit histogram index.

```
dataset = port.datasets.obtain(key=0)
```

Obtain Multiple

Obtain multiple existing histograms on the port with explicit histogram indices.

```
dataset_list = port.datasets.obtain_multiple(*[0,1,2])
```

Remove

Remove a histogram on the port with an explicit histogram index by the index manager of the port.

Corresponding CLI command: `PD_DELETE`

```
await port.datasets.remove(position_idx=0)
```

Filter

Filter APIs for Valkyrie.

Configuration

Enable

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.

Corresponding CLI command: `PF_ENABLE`

```
await filter.enable.set(on_off=enums.OnOff.ON)
await filter.enable.set_on()
await filter.enable.set(on_off=enums.OnOff.OFF)
await filter.enable.set_off()

resp = await filter.enable.get()
resp.on_off
```

Description

The description of a filter.

Corresponding CLI command: `PF_COMMENT`

```
await filter.comment.set(comment="this is a comment")

resp = await filter.comment.get()
resp.comment
```

Condition

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_xindex] = 2^match_term_xindex`

Value for length term `[length_term_xindex] = 2^(length_term_xindex+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.

Corresponding CLI command: `PF_CONDITION`

```
await filter.condition.set(
    and_expression_0=0,
    and_not_expression_0=0,
    and_expression_1=0,
    and_not_expression_1=0,
    and_expression_2=0,
    and_expression_3=0
    )

resp = await filter.condition.get()
resp.and_expression_0
resp.and_not_expression_0
resp.and_expression_1
resp.and_not_expression_1
resp.and_expression_2
resp.and_expression_3
```

String Representation

The string representation of a filter.

Corresponding CLI command: `PF_STRING`

```
await filter.string.set(string_name="this is a name")

resp = await filter.string.get()
resp.string_name
```

Create, Obtain, Remove

Create and Obtain

Create a filter on the port, and obtain the filter object. The filter index is automatically assigned by the port.

Corresponding CLI command: `PF_CREATE`

```
filter = await port.filters.create()
```

Obtain One

Obtain an existing filter on the port with an explicit filter index.

```
filter = port.filters.obtain(position_idx=0)
```

Obtain Multiple

Obtain multiple existing filters on the port with explicit filter indices.

```
filter_list = port.filters.obtain_multiple(*[0,1,2])
```

Remove

Remove a filter on the port with an explicit filter index by the index manager of the port.

Corresponding CLI command: `PF_DELETE`

```
await port.filters.remove(position_idx=0)
```

Match Term

Match Term APIs for Valkyrie.

Configuration

Match

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.

Corresponding CLI command: `PM_MATCH`

```
await match_term.match.set(mask=Hex("FF"), value=Hex("00"))

resp = await match_term.match.get()
resp.mask
resp.value
```

Position

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

Corresponding CLI command: PM_POSITION

```
await match_term.position.set(byte_offset=0)

resp = await match_term.position.get()
resp.byte_offset
```

Protocol Segments

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.

Corresponding CLI command: PM_PROTOCOL

```
await match_term.protocol.set(segments=[enums.ProtocolOption.VLAN])

resp = await match_term.protocol.get()
resp.segments
```

Create, Obtain, Remove

Create and Obtain

Create a match term on the port, and obtain the match term object. The match term index is automatically assigned by the port.

Corresponding CLI command: PM_CREATE

```
match_term = await port.match_terms.create()
```

Obtain One

Obtain an existing match term on the port with an explicit match term index.

```
match_term = port.match_terms.obtain(key=0)
```

Obtain Multiple

Obtain multiple existing match terms on the port with explicit match term indices.

```
match_term_list = port.match_terms.obtain_multiple(*[0,1,2])
```

Remove

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.

Corresponding CLI command: `PM_DELETE`

```
await port.match_terms.remove(position_idx=0)
```

Length Term

Length Term APIs for Valkyrie.

Configuration

Length

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

Corresponding CLI command: `PL_LENGTH`

```
await length_term.length.set(
    length_check_type=enums.LengthCheckType.AT_MOST,
    size=100)
await length_term.length.set(
    length_check_type=enums.LengthCheckType.AT_LEAST,
    size=100)

resp = await length_term.length.get()
resp.length_check_type
resp.size
```

Create, Obtain, Remove

Create and Obtain

Create a length term on the port, and obtain the length term object. The length term index is automatically assigned by the port.

```
length_term = await port.length_terms.create()
```

Obtain One

Obtain an existing length term on the port with an explicit length term index.

```
length_term = port.length_terms.obtain(key=0)
```

Obtain Multiple

Obtain multiple existing length terms on the port with explicit length term indices.

```
length_term_list = port.length_terms.obtain_multiple(*[0,1,2])
```

Remove

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.

Corresponding CLI command: `PL_DELETE`

```
# Remove a length term on the port with an explicit length term index␣
↪by the index manager of the port.
await port.length_terms.remove(position_idx=0)
```

PCS/PMA

PCS/PMA APIs for high-speed ports.

Auto-Negotiation

Configuration

Auto-negotiation configuration.

Corresponding CLI command: `PP_AUTONEG`

```
# Auto-Negotiation Settings
resp = await port.pcs_pma.auto_neg.settings.get()
resp.tec_ability
resp.fec_capable
resp.fec_requested
resp.pause_mode
```

Status

Status of auto-negotiation.

Corresponding CLI command: `PP_AUTONEGSTATUS`

```
a# Auto-Negotiation Status
resp = await port.pcs_pma.auto_neg.status.get()
resp.mode
resp.auto_state
resp.tec_ability
resp.fec_capable
resp.fec_requested
resp.fec
resp.pause_mode
```

Selection

Whether the port responds to incoming auto-negotiation requests.

Note: Only applicable to RJ45 ports

Corresponding CLI command: `P_AUTONEGSELECTION`

```
# Auto-Negotiation Selection
# Only applicable to RJ45 ports
await port.autoneg_selection.set(on_off=enums.OnOff.ON)
await port.autoneg_selection.set_on()
await port.autoneg_selection.set(on_off=enums.OnOff.OFF)
```

(continues on next page)

(continued from previous page)

```
await port.autoneg_selection.set_off()

resp = await port.autoneg_selection.get()
resp.on_off
```

Forward Error Correction

FEC Mode

FEC mode for port that supports FEC.

Corresponding CLI command: `PP_FECMODE`

```
# FEC Mode
await port.fec_mode.set(mode=enums.FECMode.RS_FEC)
await port.fec_mode.set(mode=enums.FECMode.RS_FEC_KP)
await port.fec_mode.set(mode=enums.FECMode.RS_FEC_KR)
await port.fec_mode.set(mode=enums.FECMode.FC_FEC)
await port.fec_mode.set(mode=enums.FECMode.OFF)
await port.fec_mode.set(mode=enums.FECMode.ON)

resp = await port.fec_mode.get()
resp.mode
```

Link Training

Configuration

Link training settings

Corresponding CLI command: `PP_LINKTRAIN`

```
# Link Training Settings
await port.pcs_pma.link_training.settings.set(
    mode=enums.LinkTrainingMode.DISABLED,
    pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
    nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
    nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
    timeout_mode=enums.TimeoutMode.DEFAULT)
await port.pcs_pma.link_training.settings.set(
    mode=enums.LinkTrainingMode.STANDALONE,
    pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
    nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
    nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
```

(continues on next page)

(continued from previous page)

```
    timeout_mode=enums.TimeoutMode.DEFAULT)
await port.pcs_pma.link_training.settings.set(
    mode=enums.LinkTrainingMode.INTERACTIVE,
    pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
    nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
    nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
    timeout_mode=enums.TimeoutMode.DISABLED)
await port.pcs_pma.link_training.settings.set(
    mode=enums.LinkTrainingMode.START_AFTER_AUTONEG,
    pam4_frame_size=enums.PAM4FrameSize.P4K_FRAME,
    nrz_pam4_init_cond=enums.LinkTrainingInitCondition.NO_INIT,
    nrz_preset=enums.NRZPreset.NRZ_WITH_PRESET,
    timeout_mode=enums.TimeoutMode.DEFAULT)

resp = await port.pcs_pma.link_training.settings.get()
resp.mode
resp.pam4_frame_size
resp.nrz_pam4_init_cond
resp.nrz_preset
resp.timeout_mode
```

Per Serdes Status

Per lane Link training status

Corresponding CLI command: `PP_LINKTRAINSTATUS`

```
# Link Training Serdes Status
resp = await port.pcs_pma.link_training.per_lane_status[0].get() #␣
↪serdes lane 0
resp.mode
resp.failure
resp.status
```

PMA Pulse Error Inject

Control

Enable / disable 'PMA pulse error inject'.

Corresponding CLI command: `PP_PMAERRPUL_ENABLE`

```
# PMA Pulse Error Inject Control
await port.pcs_pma.pma_pulse_err_inj.enable.set(on_off=enums.OnOff.ON)
```

(continues on next page)

(continued from previous page)

```
await port.pcs_pma.pma_pulse_err_inj.enable.set_on()
await port.pcs_pma.pma_pulse_err_inj.enable.set(on_off=enums.OnOff.OFF)
await port.pcs_pma.pma_pulse_err_inj.enable.set_off()

resp = await port.pcs_pma.pma_pulse_err_inj.enable.get()
resp.on_off
```

Configuration

The 'PMA pulse error inject'.

Note: Period must be > duration. BER will be: coeff * 10exp

Corresponding CLI command: `PP_PMAERRPUL_PARAMS`

```
# PMA Pulse Error Inject Configuration
await port.pcs_pma.pma_pulse_err_inj.params.set(duration=1000,␣
↪period=1000, repetition=10, coeff=5, exp=-5)

resp = await port.pcs_pma.pma_pulse_err_inj.params.get()
resp.duration
resp.period
resp.coeff
resp.exp
```

RX Status

Lane Error Counters

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

Corresponding CLI command: `PP_RXLANEERRORS`

```
# RX Status - Lane Error Counters
resp = await port.pcs_pma.lanes[0].rx_status.errors.get()
resp.alignment_error_count
resp.corrected_fec_error_count
resp.header_error_count
```

Lock Status

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

Corresponding CLI command: PP_RXLANELOCK

```
# RX Status - Lock Status
resp = await port.pcs_pma.lanes[0].rx_status.lock.get()
resp.align_lock
resp.header_lock
```

Lane Status

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.

Corresponding CLI command: PP_RXLANESTATUS

```
# RX Status - Lane Status
resp = await port.pcs_pma.lanes[0].rx_status.status.get()
resp.skew
resp.virtual_lane
```

Clear Counters

Clear all the PCS/PMA receiver statistics for a port.

Corresponding CLI command: PP_RXCLEAR

```
# RX Status - Clear Counters
await port.pcs_pma.rx.clear.set()
```

RX FEC Stats

Provides statistics on how many FEC blocks have been seen with a given number of symbol errors.

Corresponding CLI command: PP_RXFECSTATS

```
# RX Status - RX FEC Stats
resp = await port.pcs_pma.rx.fec_status.get()
resp.stats_type
resp.data_count # number of values in stats
resp.stats # list of long integers, array of length value_count. The␣
↪stats array shows how many FEC blocks have been seen with [0, 1, 2,␣
```

(continues on next page)

(continued from previous page)

```
↪3....15, >15] symbol errors and the last one is the sum of FEC␣
↪blocks with <=n symbol errors
```

RX Total Stats

Provides FEC Total counters.

Corresponding CLI command: `PP_RXTOTALSTATS`

```
# RX Status - RX Total Stats
resp = await port.pcs_pma.rx.total_status.get()
resp.total_corrected_codeword_count
resp.total_corrected_symbol_count
resp.total_rx_bit_count
resp.total_rx_codeword_count
resp.total_uncorrectable_codeword_count
post_fec_ber = 1/resp.total_post_fec_ber
pre_fec_ber = 1/resp.total_pre_fec_ber
```

TX Configuration

Error Counters

Obtain the error count of each alarm, PCS Error, FEC Error, Header Error, Align Error, BIP Error, and High BER Error.

Corresponding CLI command: `PP_ALARMS_ERRORS`

```
# TX Configuration - Error Counters
resp = await port.pcs_pma.alarms.errors.get()
resp.total_alarms
resp.los_error_count
resp.total_align_error_count
resp.total_bip_error_count
resp.total_fec_error_count
resp.total_header_error_count
resp.total_higher_error_count
resp.total_pcs_error_count
resp.valid_mask
```

Error Generation Rate

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

Corresponding CLI command: `PP_TXERRORRATE`

```
# TX Configuration - Error Generation Rate
resp = await port.pcs_pma.error_gen.error_rate.get()
resp.rate
```

Error Generation Inject

Inject a single bit-level error into the SerDes where injection has been enabled.

Corresponding CLI command: `PP_TXINJECTONE`

```
# TX Configuration - Error Generation Inject
await port.pcs_pma.error_gen.inject_one.set()
```

Error Injection

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

Corresponding CLI command: `PP_TXLANEINJECT`

```
# TX Configuration - Error Injection
await port.pcs_pma.lanes[0].tx_error_inject.set_alignerror()
await port.pcs_pma.lanes[0].tx_error_inject.set_bip8error()
await port.pcs_pma.lanes[0].tx_error_inject.set_headererror()
```

Lane Configuration

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

Corresponding CLI command: `PP_TXLANECONFIG`

```
# TX Configuration - Lane Configuration
await port.pcs_pma.lanes[0].tx_config.set(virt_lane_index=1, skew=10)

resp = await port.pcs_pma.lanes[0].tx_config.get()
resp.virt_lane_index
resp.skew
```

PHY

PHY settings for high-speed ports.

Eye Diagram

Information

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.

Corresponding CLI command: `PP_EYEINFO`

```
# Eye Diagram Information
resp = await port.serdes[0].eye_diagram.info.get()
resp.width_mui
resp.height_mv
resp.h_slope_left
resp.h_slope_right
resp.y_intercept_left
resp.y_intercept_right
resp.r_squared_fit_left
resp.r_squared_fit_right
resp.est_rj_rms_left
resp.est_rj_rms_right
resp.est_dj_pp
resp.v_slope_bottom
resp.v_slope_top
resp.x_intercept_bottom
resp.x_intercept_top
resp.r_squared_fit_bottom
resp.r_squared_fit_top
resp.est_rj_rms_bottom
resp.est_rj_rms_top
```

Bit Error Rate

Obtain BER estimations of an eye diagram.

Corresponding CLI command: `PP_EYEBER`

```
# Eye Diagram Bit Error Rate
resp = await port.serdes[0].eye_diagram.ber.get()
resp.eye_ber_estimation
```

Dwell Bits

Min and max dwell bits for an eye capture.

Corresponding CLI command: PP_EYEDWELLBITS

```
# Eye Diagram Dwell Bits
resp = await port.serdes[0].eye_diagram.dwell_bits.get()
resp.max_dwell_bit_count
resp.min_dwell_bit_count
```

Measure

Start/stop a new BER eye-measure on a 25G serdes. Use "get" to see the status of the data gathering process.

Corresponding CLI command: PP_EYEMEASURE

```
# Eye Diagram Measure
resp = await port.serdes[0].eye_diagram.measure.get()
resp.status
```

Resolution

Set or get the resolution used for the next BER eye-measurement.

Corresponding CLI command: PP_EYERESOLUTION

```
# Eye Diagram Resolution
resp = await port.serdes[0].eye_diagram.resolution.get()
resp.x_resolution
resp.y_resolution
```

Data Columns

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: 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).

Corresponding CLI command: PP_EYEREAD

```
# Eye Diagram Data Columns
resp = await port.serdes[0].eye_diagram.read_column[0].get()
resp.valid_column_count
resp.values
resp.x_resolution
resp.y_resolution
```

Settings and Status

Signal Status

Obtain the PHY signal status.

Corresponding CLI command: `PP_PHYSIGNALSTATUS`

```
# PHY - Signal Status
resp = await port.pcs_pma.phy.signal_status.get()
resp.phy_signal_status
```

Settings

Get/Set low-level PHY settings.

Corresponding CLI command: `PP_PHYSETTINGS`

```
# PHY - Settings
await port.pcs_pma.phy.settings.set(
    link_training_on_off=enums.OnOff.ON,
    precode_on_off=enums.OnOffDefault.DEFAULT,
    graycode_on_off=enums.OnOff.OFF, pam4_msb_lsb_swap=enums.OnOff.OFF)

resp = await port.pcs_pma.phy.settings.get()
resp.link_training_on_off
resp.precode_on_off
resp.graycode_on_off
resp.pam4_msb_lsb_swap
```

Tap Configuration

TX Tap Autotune

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).

Corresponding CLI command: PP_PHYAUTOTUNE

```
# TX Tap Autotune
await port.serdes[0].phy.autotune.set(on_off=enums.OnOff.ON)
await port.serdes[0].phy.autotune.set_on()
await port.serdes[0].phy.autotune.set(on_off=enums.OnOff.OFF)
await port.serdes[0].phy.autotune.set_off()

resp = await port.serdes[0].phy.autotune.get()
resp.on_off
```

TX Tap Retune

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.

Corresponding CLI command: PP_PHYRETUNE

```
# TX Tap Retune
await port.serdes[0].phy.retune.set(dummy=1)
```

TX Tap Configuration

Control and monitor the equalizer settings of the on-board PHY in the transmission direction (towards the transceiver cage) on Thor and Loki modules.

Corresponding CLI command: PP_PHYTXEQ

```
# TX Tap Configuration
await port.serdes[0].phy.tx_equalizer.set(pre2=0, pre1=0, main=86,␣
↪post1=0, post2=0, post3=0)
resp = await port.serdes[0].phy.tx_equalizer.get()
resp.pre2
resp.pre
resp.main
resp.post
```

(continues on next page)

(continued from previous page)

```
resp.pre3_post2 # pre3 for Freya (112G Serdes), post2 for Thor (56G␣
↪Serdes)
resp.post3
```

RX Tap Configuration

RX EQ parameters.

Note: For non-Freya Modules.

Corresponding CLI command: PP_PHYRXEQ

```
# RX Tap Configuration
await port.serdes[0].phy.rx_equalizer.set(auto=0, ctle=0, reserved=0)

resp = await port.serdes[0].phy.rx_equalizer.get()
resp.auto
resp.ctle
```

PRBS

PRBS settings for high-speed ports.

Configuration

Type

Defines the PRBS type used when the interface is in PRBS mode.

Corresponding CLI command: PP_PRBSTYPE

```
# PRBS Configuration
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS7,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.CAUI_VIRTUAL,
    polynomial=enums.PRBSPolynomial.PRBS9,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.PERSECOND)
```

(continues on next page)

(continued from previous page)

```
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS10,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS11,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS13,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS15,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS20,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS23,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS31,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS49,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
await port.serdes[0].prbs.config.type.set(
    prbs_inserted_type=enums.PRBSInsertedType.PHY_LINE,
    polynomial=enums.PRBSPolynomial.PRBS58,
    invert=enums.PRBSInvertState.NON_INVERTED,
    statistics_mode=enums.PRBSStatisticsMode.ACCUMULATIVE)
```

(continues on next page)

(continued from previous page)

```
resp = await port.serdes[0].prbs.config.type.get()
resp.prbs_inserted_type
resp.polynomial
resp.invert
resp.statistics_mode
```

Statistics

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

Corresponding CLI command: `PP_RXPRBSSTATUS`

```
# PRBS Statistics
resp = await port.serdes[0].prbs.status.get()
resp.byte_count
resp.error_count
resp.lock
```

Statistics

Statistics for Valkyrie ports.

Error Counter

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.

Corresponding CLI command: `P_ERRORS`

```
# Error Counter
resp = await port.errors_count.get()
resp.error_count
```

RX Statistics

Clear Counter

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

Corresponding CLI command: `PR_CLEAR`

```
# RX Statistics - Clear Counter
await port.statistics.rx.clear.set()
```

Calibrate

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.

Corresponding CLI command: `PR_CALIBRATE`

```
# RX Statistics - Calibrate
await port.statistics.rx.calibrate.set()
```

Total Counter

Obtains statistics concerning all the packets received on a port.

Corresponding CLI command: `PR_TOTAL`

```
# RX Statistics - Total Counter
resp = await port.statistics.rx.total.get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

Non-TPLD Counter

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

Corresponding CLI command: `PR_NOTPLD`

```
# RX Statistics - Non-TPLD Counter
resp = await port.statistics.rx.no_tpld.get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

PFC Counter

Obtains statistics of received Priority Flow Control (PFC) packets on a port.

Corresponding CLI command: `PR_PFCSTATS`

```
# RX Statistics - PFC Counter
resp = await port.statistics.rx.pfc_stats.get()
resp.packet_count
resp.quanta_pri_0
resp.quanta_pri_1
resp.quanta_pri_2
resp.quanta_pri_3
resp.quanta_pri_4
resp.quanta_pri_5
resp.quanta_pri_6
resp.quanta_pri_7
```

Extra Counter

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

Corresponding CLI command: `PR_EXTRA`

```
# RX Statistics - Extra Counter
resp = await port.statistics.rx.extra.get()
resp.fcs_error_count
resp.pause_frame_count
resp.gap_count
resp.gap_duration
resp.pause_frame_count
resp.rx_arp_reply_count
resp.rx_arp_request_count
resp.rx_ping_reply_count
resp.rx_ping_request_count
```

Received TPLDs

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.

Corresponding CLI command: `PR_TPLDS`

```
# RX Statistics - Received TPLDs
await port.statistics.rx.obtain_available_tplds()
```

TPLD - Error Counter

Obtains statistics concerning errors in the packets with a particular test payload id received on a port. The error information is derived from analyzing 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.

Corresponding CLI command: `PR_TPLDERRORS`

```
# RX Statistics - TPLD - Error Counter
resp = await port.statistics.rx.access_tpld(tpld_id=0).errors.get()
resp.non_incre_payload_packet_count
resp.non_incre_seq_event_count
resp.swapped_seq_misorder_event_count
```

TPLD - Latency Counter

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.

Corresponding CLI command: `PR_TPLDLATENCY`

```
# RX Statistics - TPLD - Latency Counter
resp = await port.statistics.rx.access_tpld(tpld_id=0).latency.get()
resp.avg_last_sec
resp.max_last_sec
resp.min_last_sec
resp.avg_val
resp.max_val
resp.min_val
```

TPLD - Jitter Counter

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.

Corresponding CLI command: `PR_TPLDJITTER`

```
# RX Statistics - TPLD - Jitter Counter
resp = await port.statistics.rx.access_tpld(tpld_id=0).jitter.get()
resp.avg_last_sec
resp.max_last_sec
resp.min_last_sec
resp.avg_val
resp.max_val
resp.min_val
```

TPLD - Traffic Counter

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

Corresponding CLI command: `PR_TPLDTRAFFIC`

```
# RX Statistics - TPLD - Traffic Counter
resp = await port.statistics.rx.access_tpld(tpld_id=0).traffic.get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

Filter Statistics

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

Corresponding CLI command: `PR_FILTER`

```
# RX Statistics - Filter Statistics
resp = await port.statistics.rx.obtain_filter_statistics(filter=0).
↪get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

UAT Status

This command will show the current UAT (UnAvailable Time) state, which is used in Valkyrie1564.

Corresponding CLI command: `PR_UAT_STATUS`

```
await port.statistics.rx.uat.status.get()
```

UAT Time

This command will show the current number of unavailable seconds, which is used in Valkyrie1564.

Corresponding CLI command: `PR_UAT_TIME`

```
await port.statistics.rx.uat.time.get()
```

TX Statistics

Clear Counter

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

Corresponding CLI command: `PT_CLEAR`

```
# TX Statistics - Clear Counter
await port.statistics.tx.clear.set()
```

Total Counter

Obtains statistics concerning all the packets transmitted on a port.

Corresponding CLI command: `PT_TOTAL`

```
# TX Statistics - Total Counter
resp = await port.statistics.tx.total.get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

Non-TPLD Counter

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

Corresponding CLI command: `PT_NOTPLD`

```
# TX Statistics - Non-TPLD Counter
resp = await port.statistics.tx.no_tpld.get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

Extra Counter

Obtains additional statistics for packets transmitted on a port.

Corresponding CLI command: `PT_EXTRA`

```
# TX Statistics - Extra Counter
resp = await port.statistics.tx.extra.get()
resp.tx_arp_req_count
```

Stream Counter

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

Corresponding CLI command: `PT_STREAM`

```
# TX Statistics - Stream Counter
resp = await port.statistics.tx.obtain_from_stream(stream=0).get()
resp.byte_count_since_cleared
resp.packet_count_since_cleared
resp.bit_count_last_sec
resp.packet_count_last_sec
```

Impairment

Note: Applicable to Chimera port only.

Impairment Configuration

Impairment On/OFF

The action determines if emulation functionality is enabled or disabled.

Corresponding CLI command: P_EMULATE

```
await port.emulate.set(action=enums.OnOff.ON)
await port.emulate.set(action=enums.OnOff.OFF)

resp = await port.emulate.get()
resp.action
```

FCS Error Action

The action on packets with FCS errors on a port.

Corresponding CLI command: PE_FCSDROP

```
await port.emulation.drop_fcs_errors.set(action=enums.OnOff.ON)
await port.emulation.drop_fcs_errors.set(action=enums.OnOff.OFF)

resp = await port.emulation.drop_fcs_errors.get()
resp.action
```

TPLD Mode

The action indicates the TPLD mode to be used per port.

Corresponding CLI command: PE_TPLDMODE

```
# Set TPLD mode
await port.emulation.tpld_mode.set(mode=enums.TPLDMode.NORMAL)
await port.emulation.tpld_mode.set(mode=enums.TPLDMode.MICRO)

resp = await port.emulation.tpld_mode.get()
resp.mode
```

Filter

Note: Applicable to Chimera port only.

Properties

Description

Flow description.

Corresponding CLI command: `PE_COMMENT`

```
flow = port.emulation.flows[0]
await flow.comment.set(comment="Flow description")

resp = await port.emulation.flows[0].comment.get()
resp.comment
```

Initiation

Prepares for setting up a filter definition. When called, all filter definitions in the shadow-set which are not applied are discarded and replaced with the default values (DEFAULT).

Note: 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`.

Corresponding CLI command: `PEF_INIT`

```
flow = port.emulation.flows[0]
await flow.shadow_filter.initiating.set()
```

Apply

Applies filter definitions from "shadow-copy" to "working-copy". This also pushes these settings to the FPGA.

Corresponding CLI command: `PEF_APPLY`

```
flow = port.emulation.flows[0]
await flow.shadow_filter.apply.set()
```

Enable

Defines if filtering is enabled for the flow.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_ENABLE`

```
flow = port.emulation.flows[0]
await flow.shadow_filter.enable.set(state=enums.OnOff.ON)
await flow.shadow_filter.enable.set(state=enums.OnOff.OFF)

resp = await flow.shadow_filter.enable.get()
resp.state
```

Cancel

Undo updates to shadow filter settings, sets dirty false.

Corresponding CLI command: `PEF_CANCEL`

```
flow = port.emulation.flows[0]
await flow.shadow_filter.cancel.set()
```

Filter Mode

Control the filter mode.

Corresponding CLI command: `PEF_MODE`

```
flow = port.emulation.flows[0]
await flow.shadow_filter.use_basic_mode()
await flow.shadow_filter.use_extended_mode()
```

(continues on next page)

(continued from previous page)

```
filter = await flow.shadow_filter.get_mode()

if isinstance(filter, misc.BasicImpairmentFlowFilter):
    ...

if isinstance(filter, misc.ExtendedImpairmentFlowFilter):
    ...
```

Basic Filter Configuration

Note: Applicable to Chimera port only.

Any

Note: Applicable to Chimera port only.

Configuration

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`.

Corresponding CLI command: `PEF_ANYCONFIG`

```
from xoa_driver import misc

filter = await port.emulation.flows[1].shadow_filter.get_mode() # e.g.␣
↪flow_id = 1
if isinstance(filter, misc.BasicImpairmentFlowFilter):
    await filter.any.config.set(position=0, value=Hex("112233445566"),␣
↪mask=Hex("112233445566"))

    resp = await filter.any.config.get()
    resp.position
    resp.value
    resp.mask
```

Settings

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`.

Corresponding CLI command: `PEF_ANYSETTINGS`

```
from xoa_driver import misc

filter = await port.emulation.flows[1].shadow_filter.get_mode() # e.g.␣
↪flow_id = 1
if isinstance(filter, misc.BasicImpairmentFlowFilter):
    await filter.any.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE)
    await filter.any.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.INCLUDE)

    resp = filter.any.settings.get()
    resp.use
    resp.action
```

L2

Note: Applicable to Chimera port only.

Ethernet DST

Defines the Ethernet Destination Address settings for the Ethernet filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_ETHDESTADDR`

```
await filter.ethernet.dest_address.set(use=enums.OnOff.OFF, value=Hex(
↪"BBBBBBBBBBBB"), mask=Hex("FFFFFFFFFFFF"))
await filter.ethernet.dest_address.set(use=enums.OnOff.ON, value=Hex(
↪"BBBBBBBBBBBB"), mask=Hex("FFFFFFFFFFFF"))

resp = await filter.ethernet.dest_address.get()
resp.use
```

(continues on next page)

(continued from previous page)

```
resp.value
resp.mask
```

Ethernet SRC

Defines the Ethernet Source Address settings for the Ethernet filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_ETHSRCADDR`

```
await filter.ethernet.src_address.set(use=enums.OnOff.OFF, value=Hex(
↪"AAAAAAAAAAAA"), mask=Hex("FFFFFFFFFFFF"))
await filter.ethernet.src_address.set(use=enums.OnOff.ON, value=Hex(
↪"AAAAAAAAAAAA"), mask=Hex("FFFFFFFFFFFF"))

resp = await filter.ethernet.src_address.get()
resp.use
resp.value
resp.mask
```

Ethernet Settings

Defines what filter action is performed on the Ethernet header.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_ETHSETTINGS`

```
await filter.ethernet.settings.set(use=enums.FilterUse.OFF,␣
↪action=enums.InfoAction.EXCLUDE)
await filter.ethernet.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE)
await filter.ethernet.settings.set(use=enums.FilterUse.AND,␣
↪action=enums.InfoAction.EXCLUDE)

resp = await filter.ethernet.settings.get()
resp.use
resp.action
```

L2+

Note: Applicable to Chimera port only.

Type

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`

Corresponding CLI command: `PEF_L2PUSE`

```
await filter.l2plus_use.set(use=enums.L2PlusPresent.VLAN1)
await filter.l2plus_use.set_vlan1()
await filter.l2plus_use.set(use=enums.L2PlusPresent.VLAN2)
await filter.l2plus_use.set_vlan2()
await filter.l2plus_use.set(use=enums.L2PlusPresent.MPLS)
await filter.l2plus_use.set_mpls()
await filter.l2plus_use.set(use=enums.L2PlusPresent.NA)
await filter.l2plus_use.set_na()

resp = await filter.l2plus_use.get()
resp.use
```

VLAN Inner Tag

Basic mode only. Defines the VLAN TAG settings for the VLAN filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_VLANTAG`

```
await filter.vlan.inner.tag.set(use=enums.OnOff.ON, value=1234,␣
↪mask=Hex("0FFF"))
await filter.vlan.inner.tag.set_on()
await filter.vlan.inner.tag.set(use=enums.OnOff.OFF, value=1234,␣
↪mask=Hex("0FFF"))
await filter.vlan.inner.tag.set_off()

resp = await filter.vlan.inner.tag.get()
resp.use
```

(continues on next page)

(continued from previous page)

```
resp.value
resp.mask
```

VLAN Inner PCP

Basic mode only. Defines the VLAN PCP settings for the VLAN filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_VLANPCP`

```
await filter.vlan.inner.pcp.set(use=enums.OnOff.ON, value=3, mask=Hex(
↪"07"))
await filter.vlan.inner.pcp.set_on()
await filter.vlan.inner.pcp.set(use=enums.OnOff.OFF, value=3, mask=Hex(
↪"07"))
await filter.vlan.inner.pcp.set_off()

resp = await filter.vlan.inner.pcp.get()
resp.use
resp.value
resp.mask
```

VLAN Outer Tag

Basic mode only. Defines the VLAN TAG settings for the VLAN filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_VLANTAG`

```
await filter.vlan.outer.tag.set(use=enums.OnOff.ON, value=1234,␣
↪mask=Hex("0FFF"))
await filter.vlan.outer.tag.set_on()
await filter.vlan.outer.tag.set(use=enums.OnOff.OFF, value=1234,␣
↪mask=Hex("0FFF"))
await filter.vlan.outer.tag.set_off()

resp = await filter.vlan.outer.tag.get()
resp.use
resp.value
resp.mask
```

VLAN Outer PCP

Basic mode only. Defines the VLAN PCP settings for the VLAN filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_VLANPCP`

```
await filter.vlan.outer.pcp.set(use=enums.OnOff.ON, value=3, mask=Hex(
↪"07"))
await filter.vlan.outer.pcp.set_on()
await filter.vlan.outer.pcp.set(use=enums.OnOff.OFF, value=3, mask=Hex(
↪"07"))
await filter.vlan.outer.pcp.set_off()

resp = await filter.vlan.outer.pcp.get()
resp.use
resp.value
resp.mask
```

VLAN Settings

Defines what filter action is performed on the VLAN header.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_VLANSETTINGS`

```
await filter.vlan.settings.set(use=enums.FilterUse.OFF, action=enums.
↪InfoAction.EXCLUDE)
await filter.vlan.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.EXCLUDE)
await filter.vlan.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.INCLUDE)

resp = await filter.vlan.settings.get()
resp.use
resp.action
```

MPLS Label

Basic mode only. Defines the MPLS label settings for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_MPLSLABEL`

```
await filter.mpls.label.set(use=enums.OnOff.ON, value=1000, mask=Hex(
↪"FFFFF"))
await filter.mpls.label.set(use=enums.OnOff.OFF, value=1000, mask=Hex(
↪"FFFFF"))

resp = await filter.mpls.label.get()
resp.use
resp.value
```

MPLS TOC

Basic mode only. Defines the MPLS TOC settings for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_MPLSTOC`

```
await filter.mpls.toc.set(use=enums.OnOff.ON, value=0, mask=Hex("07"))
await filter.mpls.toc.set(use=enums.OnOff.OFF, value=0, mask=Hex("07"))

resp = await filter.mpls.toc.get()
resp.use
resp.value
```

MPLS Settings

Basic mode only. Defines what filter action is performed on the MPLS header.

Corresponding CLI command: `PEF_MPLSSETTINGS`

```
await filter.mpls.settings.set(use=enums.FilterUse.OFF, action=enums.
↪InfoAction.EXCLUDE)
await filter.mpls.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.EXCLUDE)
await filter.mpls.settings.set(use=enums.FilterUse.AND, action=enums.
```

(continues on next page)

(continued from previous page)

```
↪InfoAction.INCLUDE)

resp = await filter.mpls.settings.get()
resp.use
resp.action
```

L3

Note: Applicable to Chimera port only.

Type

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`

Corresponding CLI command: `PEF_L3USE`

```
await filter.l3_use.set(use=enums.L3Present.IP4)
await filter.l3_use.set_ip4()
await filter.l3_use.set(use=enums.L3Present.IP6)
await filter.l3_use.set_ip6()
await filter.l3_use.set(use=enums.L3Present.NA)
await filter.l3_use.set_na()

resp = await filter.l3_use.get()
resp.use
```

IPv4 DST

Basic mode only. Defines the IPv4 Destination Address settings for the IPv4 filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV4DESTADDR`

```
await filter.ip.v4.dest_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF"))
await filter.ip.v4.dest_address.set(use=enums.OnOff.OFF,␣
```

(continues on next page)

(continued from previous page)

```
↪value=ipaddress.IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF"))

resp = await filter.ip.v4.dest_address.get()
resp.use
resp.value
resp.mask
```

IPv4 SRC

Basic mode only. Defines the IPv4 Source Address settings for the IPv4 filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV4SRCADDR`

```
await filter.ip.v4.src_address.set(use=enums.OnOff.ON, value=ipaddress.
↪IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF"))
await filter.ip.v4.src_address.set(use=enums.OnOff.OFF,␣
↪value=ipaddress.IPv4Address("10.0.0.2"), mask=Hex("FFFFFFFF"))

resp = await filter.ip.v4.src_address.get()
resp.use
resp.value
resp.mask
```

IPv4 DSCP

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`

Corresponding CLI command: `PEF_IPV4DSCP`

```
await filter.ip.v4.dscp.set(use=enums.OnOff.ON, value=0, mask=Hex("FC
↪"))
await filter.ip.v4.dscp.set(use=enums.OnOff.OFF, value=0, mask=Hex("FC
↪"))

resp = await filter.ip.v4.dscp.get()
resp.use
resp.value
resp.mask
```

IPv4 Settings

Basic mode only. Defines what filter action is performed on the IPv4 header.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV4SETTINGS`

```
await filter.ip.v4.settings.set(use=enums.FilterUse.OFF, action=enums.
↪InfoAction.EXCLUDE)
await filter.ip.v4.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.EXCLUDE)
await filter.ip.v4.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.INCLUDE)

resp = await filter.ip.v4.settings.get()
resp.use
resp.action
```

IPv6 DST

Basic mode only. Defines the IPv6 Destination Address settings for the IPv6 filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV6DESTADDR`

```
await filter.ip.v6.dest_address.set(use=enums.OnOff.OFF,␣
↪value=ipaddress.IPv6Address("2002::2"), mask=Hex(
↪"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))
await filter.ip.v6.dest_address.set(use=enums.OnOff.ON,␣
↪value=ipaddress.IPv6Address("2002::2"), mask=Hex(
↪"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))

resp = await filter.ip.v6.dest_address.get()
resp.use
resp.value
resp.mask
```

IPv6 SRC

Basic mode only. Defines the IPv6 Source Address settings for the IPv6 filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV6SRCADDR`

```
await filter.ip.v6.src_address.set(use=enums.OnOff.OFF,␣
↪value=ipaddress.IPv6Address("2002::2"), mask=Hex(
↪"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))
await filter.ip.v6.src_address.set(use=enums.OnOff.ON, value=ipaddress.
↪IPv6Address("2002::2"), mask=Hex("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"))

resp = await filter.ip.v6.src_address.get()
resp.use
resp.value
resp.mask
```

IPv6 Traffic Class

Basic mode only. Defines the IPv6 Traffic Class settings used for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV6TC`

```
await filter.ip.v6.traffic_class.set(use=enums.OnOff.OFF, value=0,␣
↪mask=Hex("FC"))
await filter.ip.v6.traffic_class.set(use=enums.OnOff.ON, value=0,␣
↪mask=Hex("FC"))

resp = await filter.ip.v6.traffic_class.get()
resp.use
resp.value
resp.mask
```

IPv6 Settings

Basic mode only. Defines what filter action is performed on the IPv6 header.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_IPV6SETTINGS`

```
await filter.ip.v6.settings.set(use=enums.FilterUse.OFF, action=enums.
↪InfoAction.EXCLUDE)
await filter.ip.v6.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.EXCLUDE)
await filter.ip.v6.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.INCLUDE)

resp = await filter.ip.v6.settings.get()
resp.use
resp.action
```

L4

Note: Applicable to Chimera port only.

TCP DST Port

Basic mode only. Defines TCP Destination Port settings used for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`.

Corresponding CLI command: `PEF_TCPDESTPORT`

```
await filter.tcp.dest_port.set(use=enums.OnOff.OFF, value=80, mask=Hex(
↪"FFFF"))
await filter.tcp.dest_port.set(use=enums.OnOff.ON, value=80, mask=Hex(
↪"FFFF"))

resp = await filter.tcp.dest_port.get()
resp.use
resp.value
resp.mask
```

TCP SRC Port

Basic mode only. Defines TCP Source Port settings used for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`.

Corresponding CLI command: `PEF_TCPSRCPORT`

```
await filter.tcp.src_port.set(use=enums.OnOff.OFF, value=80, mask=Hex(
 ↪"FFFF"))
await filter.tcp.src_port.set(use=enums.OnOff.ON, value=80, mask=Hex(
 ↪"FFFF"))

resp = await filter.tcp.src_port.get()
resp.use
resp.value
resp.mask
```

TCP Settings

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`.

Corresponding CLI command: `PEF_TCPSETTINGS`

```
await filter.tcp.settings.set(use=enums.FilterUse.OFF, action=enums.
 ↪InfoAction.EXCLUDE)
await filter.tcp.settings.set(use=enums.FilterUse.AND, action=enums.
 ↪InfoAction.EXCLUDE)
await filter.tcp.settings.set(use=enums.FilterUse.AND, action=enums.
 ↪InfoAction.INCLUDE)

resp = await filter.tcp.settings.get()
resp.use
resp.action
```

UDP DST Port

Basic mode only. Defines UDP Destination Port settings used for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`.

Corresponding CLI command: `PEF_UDPDESTPORT`

```
await filter.tcp.dest_port.set(use=enums.OnOff.ON, value=80, mask=Hex(
↪"FFFF"))
await filter.tcp.dest_port.set(use=enums.OnOff.OFF, value=80, mask=Hex(
↪"FFFF"))

resp = await filter.udp.dest_port.get()
resp.use
resp.value
resp.mask
```

UDP SRC Port

Basic mode only. Defines UDP Source Port settings used for the filter.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_UDPSRCPORT`

```
await filter.tcp.src_port.set(use=enums.OnOff.ON, value=80, mask=Hex(
↪"FFFF"))
await filter.tcp.src_port.set(use=enums.OnOff.OFF, value=80, mask=Hex(
↪"FFFF"))

resp = await filter.udp.src_port.get()
resp.use
resp.value
resp.mask
```

UDP Settings

Basic mode only. Controls if UDP packet information is used for flow filtering.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`

Corresponding CLI command: `PEF_UDPSETTINGS`

```
await filter.udp.settings.set(use=enums.FilterUse.OFF, action=enums.
↪InfoAction.EXCLUDE)
await filter.udp.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.EXCLUDE)
await filter.udp.settings.set(use=enums.FilterUse.AND, action=enums.
↪InfoAction.INCLUDE)

resp = await filter.udp.settings.get()
resp.use
resp.action
```

TPLD

Note: Applicable to Chimera port only.

TPLD ID Configuration

Defines the TPLD filter configuration. There are only 16 TPLD filter, thus the index values are from 0 to 15.

Note: For SET, the only allowed `_filter_type` is `shadow-copy`.

Corresponding CLI command: `PEF_TPLDCONFIG`

```
await filter.tpld.test_payload_filters_config[0].set(use=enums.OnOff.
↪ON, id = 2)
await filter.tpld.test_payload_filters_config[0].set(use=enums.OnOff.
↪OFF, id = 2)
await filter.tpld.test_payload_filters_config[1].set(use=enums.OnOff.
↪ON, id = 4)
await filter.tpld.test_payload_filters_config[1].set(use=enums.OnOff.
↪OFF, id = 4)
await filter.tpld.test_payload_filters_config[2].set(use=enums.OnOff.
```

(continues on next page)

(continued from previous page)

```
↪ON, id = 6)
await filter.tpld.test_payload_filters_config[2].set(use=enums.OnOff.
↪OFF, id = 6)
await filter.tpld.test_payload_filters_config[3].set(use=enums.OnOff.
↪ON, id = 8)
await filter.tpld.test_payload_filters_config[3].set(use=enums.OnOff.
↪OFF, id = 8)
await filter.tpld.test_payload_filters_config[4].set(use=enums.OnOff.
↪ON, id = 10)
await filter.tpld.test_payload_filters_config[4].set(use=enums.OnOff.
↪OFF, id = 10)
await filter.tpld.test_payload_filters_config[5].set(use=enums.OnOff.
↪ON, id = 20)
await filter.tpld.test_payload_filters_config[5].set(use=enums.OnOff.
↪OFF, id = 20)
await filter.tpld.test_payload_filters_config[6].set(use=enums.OnOff.
↪ON, id = 40)
await filter.tpld.test_payload_filters_config[6].set(use=enums.OnOff.
↪OFF, id = 40)
await filter.tpld.test_payload_filters_config[7].set(use=enums.OnOff.
↪ON, id = 60)
await filter.tpld.test_payload_filters_config[7].set(use=enums.OnOff.
↪OFF, id = 60)
await filter.tpld.test_payload_filters_config[8].set(use=enums.OnOff.
↪ON, id = 80)
await filter.tpld.test_payload_filters_config[8].set(use=enums.OnOff.
↪OFF, id = 80)
await filter.tpld.test_payload_filters_config[9].set(use=enums.OnOff.
↪ON, id = 100)
await filter.tpld.test_payload_filters_config[9].set(use=enums.OnOff.
↪OFF, id = 100)
await filter.tpld.test_payload_filters_config[10].set(use=enums.OnOff.
↪ON, id = 102)
await filter.tpld.test_payload_filters_config[10].set(use=enums.OnOff.
↪OFF, id = 102)
await filter.tpld.test_payload_filters_config[11].set(use=enums.OnOff.
↪ON, id = 104)
await filter.tpld.test_payload_filters_config[11].set(use=enums.OnOff.
↪OFF, id = 104)
await filter.tpld.test_payload_filters_config[12].set(use=enums.OnOff.
↪ON, id = 106)
await filter.tpld.test_payload_filters_config[12].set(use=enums.OnOff.
↪OFF, id = 106)
await filter.tpld.test_payload_filters_config[13].set(use=enums.OnOff.
↪ON, id = 108)
await filter.tpld.test_payload_filters_config[13].set(use=enums.OnOff.
```

(continues on next page)

(continued from previous page)

```
↪OFF, id = 108)
await filter.tpld.test_payload_filters_config[14].set(use=enums.OnOff.
↪ON, id = 110)
await filter.tpld.test_payload_filters_config[14].set(use=enums.OnOff.
↪OFF, id = 110)
await filter.tpld.test_payload_filters_config[15].set(use=enums.OnOff.
↪ON, id = 200)
await filter.tpld.test_payload_filters_config[15].set(use=enums.OnOff.
↪OFF, id = 200)

resp = await filter.tpld.test_payload_filters_config[0].get()
resp.use
resp.id
```

Settings

Defines if filtering on TPLD field in a packet is used for flow filtering. The TPLD filter allows filtering based on the Xena TPLD ID. The TPLD 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`.

Corresponding CLI command: `PEF_TPLDSETTINGS`

```
await filter.tpld.settings.set(action=enums.InfoAction.EXCLUDE)
await filter.tpld.settings.set(action=enums.InfoAction.INCLUDE)

resp = await filter.tpld.settings.get()
resp.action
```

Working Filter

Note: Applicable to Chimera port only.

Use Segments

This command is valid only for `Extended filter mode` (check PEF_MODE).

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).

Corresponding CLI command: `PEF_PROTOCOL`

```
# Configure shadow filter to EXTENDED mode
await flow.shadow_filter.use_extended_mode()

# Query the mode of the filter (either basic or extended)
filter = await flow.shadow_filter.get_mode()

if isinstance(filter, misc.ExtendedImpairmentFlowFilter):
    # Ethernet is the default mandatory
    # Adding VLAN after Ethernet
    await filter.use_segments(
        enums.ProtocolOption.VLAN
        )
    protocol_segments = await filter.get_protocol_segments()

    await protocol_segments[0].value.set(value=Hex(
↪"AAAAAAAAAAAABBBBBBBBBBBB8100"))
    await protocol_segments[0].mask.set(masks=Hex(
↪"00000000000000000000000000000000"))
    await protocol_segments[1].value.set(value=Hex("0064FFFF"))
    await protocol_segments[1].mask.set(masks=Hex("00000000"))
```

Segment Value

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.

Corresponding CLI command: `PEF_VALUE`

```
# Configure shadow filter to EXTENDED mode
await flow.shadow_filter.use_extended_mode()

# Query the mode of the filter (either basic or extended)
filter = await flow.shadow_filter.get_mode()

if isinstance(filter, misc.ExtendedImpairmentFlowFilter):
    # Ethernet is the default mandatory
    # Adding VLAN after Ethernet
    await filter.use_segments(
        enums.ProtocolOption.VLAN
        )
    protocol_segments = await filter.get_protocol_segments()

    await protocol_segments[0].value.set(value=Hex(
↪"AAAAAAAAAAAABBBBBBBBBBBB8100"))
    await protocol_segments[0].mask.set(masks=Hex(
↪"00000000000000000000000000000000"))
    await protocol_segments[1].value.set(value=Hex("0064FFFF"))
    await protocol_segments[1].mask.set(masks=Hex("00000000"))

    resp = await protocol_segments[0].value.get()
    resp.value
    resp = await protocol_segments[1].value.get()
    resp.value
```

Segment Mask

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.

`get/set` semantics are similar to PEF_VALUE.

Corresponding CLI command: `PEF_MASK`

```
# Configure shadow filter to EXTENDED mode
await flow.shadow_filter.use_extended_mode()

# Query the mode of the filter (either basic or extended)
filter = await flow.shadow_filter.get_mode()

if isinstance(filter, misc.ExtendedImpairmentFlowFilter):
    # Ethernet is the default mandatory
    # Adding VLAN after Ethernet
    await filter.use_segments(
        enums.ProtocolOption.VLAN
        )
    protocol_segments = await filter.get_protocol_segments()

    await protocol_segments[0].value.set(value=Hex(
↪"AAAAAAAAAAAABBBBBBBBBBBB8100"))
    await protocol_segments[0].mask.set(masks=Hex(
↪"00000000000000000000000000000000"))
    await protocol_segments[1].value.set(value=Hex("0064FFFF"))
    await protocol_segments[1].mask.set(masks=Hex("00000000"))

    resp = await protocol_segments[0].mask.get()
    resp.value
    resp = await protocol_segments[1].mask.get()
    resp.value
```

Drop

Note: Applicable to Chimera port only.

Drop Distribution

Enable/Disable

Control whether this impairment distribution is enabled.

Note: This command is not applicable for PE_BANDPOLICER and PE_BANDSHAPER because they have a separate `ON / OFF` parameter.

Corresponding CLI command: `PED_ENABLE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.enable.
↪set(action=enums.OnOff.ON)
await flow.impairment_distribution.drop_type_config.enable.set_on()
await flow.impairment_distribution.drop_type_config.enable.
↪set(action=enums.OnOff.OFF)
await flow.impairment_distribution.drop_type_config.enable.set_off()

resp = await flow.impairment_distribution.drop_type_config.off.get()
resp.action
```

Off Distribution

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.

Corresponding CLI command: `PED_OFF`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.off.set()

resp = await flow.impairment_distribution.drop_type_config.off.get()
resp.action
```

Fixed Rate Distribution

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.

Corresponding CLI command: `PED_FIXED`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.fixed_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.drop_type_config.fixed_rate.
↪get()
resp.probability
```

Random Rate Distribution

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%)

Corresponding CLI command: `PED_RANDOM`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.random_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.drop_type_config.random_rate.
↪get()
resp.probability
```

Bit Error Rate Distribution

Configuration of Bit Error Rate distribution.

Corresponding CLI command: `PED_BER`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.bit_error_rate.
↪set(coef=1, exp=1)

resp = await flow.impairment_distribution.drop_type_config.bit_error_
```

(continues on next page)

(continued from previous page)

```
↪rate.get()
resp.coef
resp.exp
```

Fixed Burst Distribution

Configuration of Fixed Burst distribution.

Corresponding CLI command: PED_FIXEDBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.fixed_burst.
↪set(burst_size=1300)

resp = await flow.impairment_distribution.drop_type_config.fixed_burst.
↪get()
resp.burst_size
```

Random Burst Distribution

Configuration of Random Burst distribution.

Corresponding CLI command: PED_RANDOMBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.random_burst.
↪set(minimum=1, maximum=10, probability=10_000)

resp = await flow.impairment_distribution.drop_type_config.random_
↪burst.get()
resp.minimum
resp.maximum
resp.probability
```

Gilbert Elliott Distribution

Configuration of Gilbert-Elliot distribution.

Corresponding CLI command: PED_GE

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.ge.set(good_state_
↪prob=0, good_state_trans_prob=0, bad_state_prob=0, bad_state_trans_
↪prob=0)
```

(continues on next page)

(continued from previous page)

```
resp = await flow.impairment_distribution.drop_type_config.ge.get()
resp.good_state_prob
resp.good_state_trans_prob
resp.bad_state_prob
resp.bad_state_trans_prob
```

Uniform Distribution

Configuration of Uniform distribution.

Note: If minimum is less than minimum, value is set to minimum. If minimum is greater than maximum, value is set to maximum.

Corresponding CLI command: `PED_UNI`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.uniform.
↪set(minimum=1, maximum=1)

resp = await flow.impairment_distribution.drop_type_config.uniform.
↪get()
resp.minimum
resp.maximum
```

Gaussian Distribution

Configuration of Gaussian distribution.

Note:

In case of `_impairment_type_xindex != 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_xindex = 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.

Corresponding CLI command: PED_GAUSS

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.gaussian.
↪set(mean=1, std_deviation=1)

resp = await flow.impairment_distribution.drop_type_config.gaussian.
↪get()
resp.mean
resp.std_deviation
```

Poisson Distribution

Configuration of "Poisson" distribution.

Note: Standard deviation is derived from mean, i.e., standard deviation = SQRT(mean).

In case of _impairment_type_xindex != DELAY, mean plus 3 times standard deviation should be less than or equal to max allowed (4194288).

In case of _impairment_type_xindex = DELAY, mean plus 3 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: PED_POISSON

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.poisson.
↪set(mean=100)

resp = await flow.impairment_distribution.drop_type_config.poisson.
↪get()
resp.mean
```

Gamma Distribution

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_xindex != 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_xindex = DELAY, mean plus 4 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: PED_GAMMA

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.gamma.set(shape=1,␣
↪scale=1)

resp = await flow.impairment_distribution.drop_type_config.gamma.get()
resp.shape
resp.scale
```

Custom Distribution

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_xindex != 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_xindex = 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.

Corresponding CLI command: PED_CUST

```
# Custom distribution for impairment Corruption
flow = port.emulation.flows[1] # e.g. flow_id = 1
data_x=[0, 1] * 256
await port.custom_distributions.assign(0)
await port.custom_distributions[0].comment.set(comment="Example Custom␣
↪Distribution")
await port.custom_distributions[0].definition.set(linear=enums.OnOff.
↪OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_
↪x)
await flow.impairment_distribution.drop_type_config.custom.set(cust_
↪id=0)

resp = await flow.impairment_distribution.drop_type_config.custom.get()
resp.cust_id
```

Drop Scheduling

Schedule

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: `PED_SCHEDULE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.drop_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.drop_type_config.schedule.
↪get()
```

One-Shot Status

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).

Corresponding CLI command: `PED_ONESHOTSTATUS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.impairment_distribution.drop_type_config.one_shot_
↪status.get()
resp.one_shot_status
```

Misordering

Note: Applicable to Chimera port only.

Misorder Configuration

Misorder Depth

Configures the misordering depth in number of packets.

Note: probability * (depth + 1) should be less than 1,000,000. (see PED_FIXED)

Corresponding CLI command: `PE_MISORDER`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.misordering.set(depth=1)

resp = await flow.misordering.get()
resp.depth
```

Misorder Distribution

Enable/Disable

Control whether this impairment distribution is enabled.

Note: This command is not applicable for PE_BANDPOLICER and PE_BANDSHAPER because they have a separate `ON / OFF` parameter.

Corresponding CLI command: `PED_ENABLE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.misorder_type_config.enable.
↪set(action=enums.OnOff.ON)
await flow.impairment_distribution.misorder_type_config.enable.set_on()
await flow.impairment_distribution.misorder_type_config.enable.
↪set(action=enums.OnOff.OFF)
await flow.impairment_distribution.misorder_type_config.enable.set_
↪off()
```

(continues on next page)

(continued from previous page)

```
resp = await flow.impairment_distribution.misorder_type_config.off.
↪get()
resp.action
```

Off Distribution

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.

Corresponding CLI command: `PED_OFF`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.misorder_type_config.off.set()

resp = await flow.impairment_distribution.misorder_type_config.off.
↪get()
resp.action
```

Fixed Rate Distribution

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.

Corresponding CLI command: `PED_FIXED`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.misorder_type_config.fixed_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.misorder_type_config.fixed_
↪rate.get()
resp.probability
```

Fixed Burst Distribution

Configuration of Fixed Burst distribution.

Corresponding CLI command: PED_FIXEDBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.misorder_type_config.fixed_burst.
↪set(burst_size=1300)

resp = await flow.impairment_distribution.misorder_type_config.fixed_
↪burst.get()
resp.burst_size
```

Misorder Scheduling

Schedule

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: PED_SCHEDULE

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.misorder_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.misorder_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.misorder_type_config.
↪schedule.get()
```

One-Shot Status

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).

Corresponding CLI command: `PED_ONESHOTSTATUS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.impairment_distribution.misorder_type_config.one_
↪shot_status.get()
resp.one_shot_status
```

Latency/Jitter

Note: Applicable to Chimera port only.

Latency & Jitter Configuration

Latency Range

Retrieve minimum and maximum configurable latency per flow in nanoseconds.

Corresponding CLI command: `PE_LATENCYRANGE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.latency_range.get()
resp.min
resp.max
```

Latency & Jitter Distribution

Enable/Disable

Control whether this impairment distribution is enabled.

Note: This command is not applicable for PE_BANDPOLICER and PE_BANDSHAPER because they have a separate `ON / OFF` parameter.

Corresponding CLI command: PED_ENABLE

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.enable.
↪set(action=enums.OnOff.ON)
await flow.impairment_distribution.latency_jitter_type_config.enable.
↪set_on()
await flow.impairment_distribution.latency_jitter_type_config.enable.
↪set(action=enums.OnOff.OFF)
await flow.impairment_distribution.latency_jitter_type_config.enable.
↪set_off()

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪off.get()
resp.action
```

Off Distribution

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.

Corresponding CLI command: PED_OFF

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.off.set()

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪off.get()
resp.action
```

Constant

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.

Corresponding CLI command: PED_CONST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.constant_
↪delay.set(delay=100)
```

(continues on next page)

(continued from previous page)

```
resp = await flow.impairment_distribution.latency_jitter_type_config.
↪constant_delay.get()
resp.delay
```

Accumulative Burst Distribution

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.

Corresponding CLI command: `PED_ACCBURST`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.
↪accumulate_and_burst.set(delay=1300)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪accumulate_and_burst.get()
resp.delay
```

Step Distribution

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.

Corresponding CLI command: `PED_STEP`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.step.
↪set(low=1300, high=77000)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪step.get()
resp.low
resp.high
```

Uniform Distribution

Configuration of Uniform distribution.

Note: If minimum is less than minimum, value is set to minimum. If minimum is greater than maximum, value is set to maximum.

Corresponding CLI command: `PED_UNI`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.uniform.
↪set(minimum=1, maximum=1)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪uniform.get()
resp.minimum
resp.maximum
```

Gaussian Distribution

Configuration of Gaussian distribution.

Note:

In case of `_impairment_type_xindex != 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_xindex = 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.

Corresponding CLI command: `PED_GAUSS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.gaussian.
↪set(mean=1, std_deviation=1)

resp = await flow.impairment_distribution.latency_jitter_type_config.
```

(continues on next page)

(continued from previous page)

```
↪gaussian.get()
resp.mean
resp.std_deviation
```

Poisson Distribution

Configuration of "Poisson" distribution.

Note: Standard deviation is derived from mean, i.e., standard deviation = SQRT(mean).

In case of `_impairment_type_xindex != DELAY`, mean plus 3 times standard deviation should be less than or equal to max allowed (4194288).

In case of `_impairment_type_xindex = DELAY`, mean plus 3 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: `PED_POISSON`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.poisson.
↪set(mean=100)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪poisson.get()
resp.mean
```

Gamma Distribution

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_xindex != 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_xindex = DELAY`, mean plus 4 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: `PED_GAMMA`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.gamma.
↪set(shape=1, scale=1)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪gamma.get()
resp.shape
resp.scale
```

Custom Distribution

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_xindex != 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_xindex = 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.

Corresponding CLI command: `PED_CUST`

```
# Custom distribution for impairment Corruption
flow = port.emulation.flows[1] # e.g. flow_id = 1
data_x=[0, 1] * 256
await port.custom_distributions.assign(0)
await port.custom_distributions[0].comment.set(comment="Example Custom␣
↪Distribution")
await port.custom_distributions[0].definition.set(linear=enums.OnOff.
↪OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_
↪x)
await flow.impairment_distribution.latency_jitter_type_config.custom.
↪set(cust_id=0)

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪custom.get()
resp.cust_id
```

Latency & Jitter Scheduling

Schedule

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: `PED_SCHEDULE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.latency_jitter_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.latency_jitter_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.latency_jitter_type_config.
↪schedule.get()
```

One-Shot Status

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).

Corresponding CLI command: `PED_ONESHOTSTATUS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.impairment_distribution.latency_jitter_type_config.
↪one_shot_status.get()
resp.one_shot_status
```

Duplication

Note: Applicable to Chimera port only.

Duplication Distribution

Enable/Disable

Control whether this impairment distribution is enabled.

Note: This command is not applicable for PE_BANDPOLICER and PE_BANDSHAPER because they have a separate `ON / OFF` parameter.

Corresponding CLI command: `PED_ENABLE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.enable.
↪set(action=enums.OnOff.ON)
await flow.impairment_distribution.duplication_type_config.enable.set_
↪on()
await flow.impairment_distribution.duplication_type_config.enable.
↪set(action=enums.OnOff.OFF)
await flow.impairment_distribution.duplication_type_config.enable.set_
↪off()

resp = await flow.impairment_distribution.duplication_type_config.off.
↪get()
resp.action
```

Off Distribution

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.

Corresponding CLI command: `PED_OFF`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.off.set()

resp = await flow.impairment_distribution.duplication_type_config.off.
↪get()
resp.action
```

Fixed Rate Distribution

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.

Corresponding CLI command: `PED_FIXED`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.fixed_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.duplication_type_config.
↪fixed_rate.get()
resp.probability
```

Random Rate Distribution

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%)

Corresponding CLI command: `PED_RANDOM`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.random_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.duplication_type_config.
↪random_rate.get()
resp.probability
```

Bit Error Rate Distribution

Configuration of Bit Error Rate distribution.

Corresponding CLI command: `PED_BER`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.bit_error_
↪rate.set(coef=1, exp=1)

resp = await flow.impairment_distribution.duplication_type_config.bit_
```

(continues on next page)

(continued from previous page)

```
↪error_rate.get()
resp.coef
resp.exp
```

Fixed Burst Distribution

Configuration of Fixed Burst distribution.

Corresponding CLI command: PED_FIXEDBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.fixed_burst.
↪set(burst_size=1300)

resp = await flow.impairment_distribution.duplication_type_config.
↪fixed_burst.get()
resp.burst_size
```

Random Burst Distribution

Configuration of Random Burst distribution.

Corresponding CLI command: PED_RANDOMBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.random_
↪burst.set(minimum=1, maximum=10, probability=10_000)

resp = await flow.impairment_distribution.duplication_type_config.
↪random_burst.get()
resp.minimum
resp.maximum
resp.probability
```

Gilbert Elliott Distribution

Configuration of Gilbert-Elliot distribution.

Corresponding CLI command: PED_GE

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.ge.set(good_
↪state_prob=0, good_state_trans_prob=0, bad_state_prob=0, bad_state_
↪trans_prob=0)
```

(continues on next page)

(continued from previous page)

```
resp = await flow.impairment_distribution.duplication_type_config.ge.
↪get()
resp.good_state_prob
resp.good_state_trans_prob
resp.bad_state_prob
resp.bad_state_trans_prob
```

Uniform Distribution

Configuration of Uniform distribution.

Note: If minimum is less than minimum, value is set to minimum. If minimum is greater than maximum, value is set to maximum.

Corresponding CLI command: `PED_UNI`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.uniform.
↪set(minimum=1, maximum=1)

resp = await flow.impairment_distribution.duplication_type_config.
↪uniform.get()
resp.minimum
resp.maximum
```

Gaussian Distribution

Configuration of Gaussian distribution.

Note:

In case of `_impairment_type_xindex != 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_xindex = 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.

Corresponding CLI command: `PED_GAUSS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.gaussian.
↪set(mean=1, std_deviation=1)

resp = await flow.impairment_distribution.duplication_type_config.
↪gaussian.get()
resp.mean
resp.std_deviation
```

Poisson Distribution

Configuration of "Poisson" distribution.

Note: Standard deviation is derived from mean, i.e., standard deviation = SQRT(mean).

In case of `_impairment_type_xindex != DELAY`, mean plus 3 times standard deviation should be less than or equal to max allowed (4194288).

In case of `_impairment_type_xindex = DELAY`, mean plus 3 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: `PED_POISSON`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.poisson.
↪set(mean=100)

resp = await flow.impairment_distribution.duplication_type_config.
↪poisson.get()
resp.mean
```

Gamma Distribution

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_xindex != 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_xindex = DELAY, mean plus 4 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: PED_GAMMA

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.gamma.
↪set(shape=1, scale=1)

resp = await flow.impairment_distribution.duplication_type_config.
↪gamma.get()
resp.shape
resp.scale
```

Custom Distribution

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_xindex != 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_xindex = 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.

Corresponding CLI command: PED_CUST

```
# Custom distribution for impairment Corruption
flow = port.emulation.flows[1] # e.g. flow_id = 1
data_x=[0, 1] * 256
await port.custom_distributions.assign(0)
await port.custom_distributions[0].comment.set(comment="Example Custom␣
↪Distribution")
await port.custom_distributions[0].definition.set(linear=enums.OnOff.
↪OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_
↪x)
await flow.impairment_distribution.duplication_type_config.custom.
↪set(cust_id=0)

resp = await flow.impairment_distribution.duplication_type_config.
```

(continues on next page)

(continued from previous page)

```
↪custom.get()
resp.cust_id
```

Duplication Scheduling

Schedule

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: `PED_SCHEDULE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.duplication_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.duplication_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.duplication_type_config.
↪schedule.get()
```

One-Shot Status

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).

Corresponding CLI command: `PED_ONESHOTSTATUS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.impairment_distribution.duplication_type_config.one_
↪shot_status.get()
resp.one_shot_status
```

Corruption

Note: Applicable to Chimera port only.

Corruption Configuration

Type

Configures impairment corruption type.

Note: IP / TCP / UDP corruption modes are not supported on default flow (0)

Corresponding CLI command: `PE_CORRUPT`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.corruption.set(corruption_type=enums.CorruptionType.OFF)
await flow.corruption.set(corruption_type=enums.CorruptionType.ETH)
await flow.corruption.set(corruption_type=enums.CorruptionType.IP)
await flow.corruption.set(corruption_type=enums.CorruptionType.TCP)
await flow.corruption.set(corruption_type=enums.CorruptionType.UDP)
await flow.corruption.set(corruption_type=enums.CorruptionType.BER)

resp = await flow.corruption.get()
resp.corruption_type
```

Corruption Distribution

Enable/Disable

Control whether this impairment distribution is enabled.

Note: This command is not applicable for PE_BANDPOLICER and PE_BANDSHAPER because they have a separate `ON / OFF` parameter.

Corresponding CLI command: `PED_ENABLE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.enable.
↪set(action=enums.OnOff.ON)
await flow.impairment_distribution.corruption_type_config.enable.set_
```

(continues on next page)

(continued from previous page)

```
↪on()
await flow.impairment_distribution.corruption_type_config.enable.
↪set(action=enums.OnOff.OFF)
await flow.impairment_distribution.corruption_type_config.enable.set_
↪off()

resp = await flow.impairment_distribution.corruption_type_config.off.
↪get()
resp.action
```

Off Distribution

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.

Corresponding CLI command: `PED_OFF`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.off.set()

resp = await flow.impairment_distribution.corruption_type_config.off.
↪get()
resp.action
```

Fixed Rate Distribution

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.

Corresponding CLI command: `PED_FIXED`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.fixed_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.corruption_type_config.fixed_
↪rate.get()
resp.probability
```

Random Rate Distribution

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%)

Corresponding CLI command: PED_RANDOM

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.random_rate.
↪set(probability=10_000)

resp = await flow.impairment_distribution.corruption_type_config.
↪random_rate.get()
resp.probability
```

Bit Error Rate Distribution

Configuration of Bit Error Rate distribution.

Corresponding CLI command: PED_BER

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.bit_error_
↪rate.set(coef=1, exp=1)

resp = await flow.impairment_distribution.corruption_type_config.bit_
↪error_rate.get()
resp.coef
resp.exp
```

Fixed Burst Distribution

Configuration of Fixed Burst distribution.

Corresponding CLI command: PED_FIXEDBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.fixed_burst.
↪set(burst_size=1300)

resp = await flow.impairment_distribution.corruption_type_config.fixed_
↪burst.get()
resp.burst_size
```

Random Burst Distribution

Configuration of Random Burst distribution.

Corresponding CLI command: PED_RANDOMBURST

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.random_burst.
↪set(minimum=1, maximum=10, probability=10_000)

resp = await flow.impairment_distribution.corruption_type_config.
↪random_burst.get()
resp.minimum
resp.maximum
resp.probability
```

Gilbert Elliott Distribution

Configuration of Gilbert-Elliot distribution.

Corresponding CLI command: PED_GE

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.ge.set(good_
↪state_prob=0, good_state_trans_prob=0, bad_state_prob=0, bad_state_
↪trans_prob=0)

resp = await flow.impairment_distribution.corruption_type_config.ge.
↪get()
resp.good_state_prob
resp.good_state_trans_prob
resp.bad_state_prob
resp.bad_state_trans_prob
```

Uniform Distribution

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.

Corresponding CLI command: PED_UNI

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.uniform.
```

(continues on next page)

(continued from previous page)

```
↪set(minimum=1, maximum=1)

resp = await flow.impairment_distribution.corruption_type_config.
↪uniform.get()
resp.minimum
resp.maximum
```

Gaussian Distribution

Configuration of Gaussian distribution.

Note:

In case of `_impairment_type_xindex != 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_xindex = 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.

Corresponding CLI command: `PED_GAUSS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.gaussian.
↪set(mean=1, std_deviation=1)

resp = await flow.impairment_distribution.corruption_type_config.
↪gaussian.get()
resp.mean
resp.std_deviation
```

Poisson Distribution

Configuration of "Poisson" distribution.

Note: Standard deviation is derived from mean, i.e., standard deviation = SQRT(mean).

In case of `_impairment_type_xindex != DELAY`, mean plus 3 times standard deviation should be less than or equal to max allowed (4194288).

In case of `_impairment_type_xindex = DELAY`, mean plus 3 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: `PED_POISSON`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.poisson.
↪set(mean=100)

resp = await flow.impairment_distribution.corruption_type_config.
↪poisson.get()
resp.mean
```

Gamma Distribution

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_xindex != 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_xindex = DELAY`, mean plus 4 times standard deviation should be less than or equal to the maximum latency.

Corresponding CLI command: `PED_GAMMA`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.gamma.
↪set(shape=1, scale=1)

resp = await flow.impairment_distribution.corruption_type_config.gamma.
↪get()
```

(continues on next page)

(continued from previous page)

```
resp.shape
resp.scale
```

Custom Distribution

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_xindex != 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_xindex = 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.

Corresponding CLI command: `PED_CUST`

```
# Custom distribution for impairment Corruption
flow = port.emulation.flows[1] # e.g. flow_id = 1
data_x=[0, 1] * 256
await port.custom_distributions.assign(0)
await port.custom_distributions[0].comment.set(comment="Example Custom␣
↪Distribution")
await port.custom_distributions[0].definition.set(linear=enums.OnOff.
↪OFF, symmetric=enums.OnOff.OFF, entry_count=len(data_x), data_x=data_
↪x)
await flow.impairment_distribution.corruption_type_config.custom.
↪set(cust_id=0)

resp = await flow.impairment_distribution.corruption_type_config.
↪custom.get()
resp.cust_id
```

Scheduling

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: `PED_SCHEDULE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.corruption_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.corruption_type_config.
↪schedule.get()

await flow.impairment_distribution.corruption_type_config.one_shot_
↪status.get()
```

Corruption Scheduling

Schedule

Configure the impairment scheduler function. The configuration of the scheduler depends on the kind 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.

Corresponding CLI command: `PED_SCHEDULE`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.impairment_distribution.corruption_type_config.schedule.
↪set(duration=1, period=1) # repeat pattern
await flow.impairment_distribution.corruption_type_config.schedule.
↪set(duration=0, period=0) # continuous

resp = await flow.impairment_distribution.corruption_type_config.
↪schedule.get()
```

One-Shot Status

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).

Corresponding CLI command: `PED_ONESHOTSTATUS`

```
flow = port.emulation.flows[1] # e.g. flow_id = 1
resp = await flow.impairment_distribution.corruption_type_config.one_
↪shot_status.get()
resp.one_shot_status
```

Policer

Note: Applicable to Chimera port only.

Policer Configuration

Configures the bandwidth policer.

Corresponding CLI command: `PE_BANDPOLICER`

```
# Configure bandwidth control - Policer
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.bandwidth_control.policer.set(on_off=enums.OnOff.ON,␣
↪mode=enums.PolicerMode.L1, cir=10_000, cbs=1_000)
await flow.bandwidth_control.policer.set(on_off=enums.OnOff.ON,␣
↪mode=enums.PolicerMode.L2, cir=10_000, cbs=1_000)

resp = await flow.bandwidth_control.policer.get()
resp.on_off
resp.mode
resp.cir
resp.cbs
```

Shaper

Note: Applicable to Chimera port only.

Shaper Configuration

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)

Corresponding CLI command: `PE_BANDSHAPER`

```
# Configure bandwidth control - Shaper
flow = port.emulation.flows[1] # e.g. flow_id = 1
await flow.bandwidth_control.shaper.set(on_off=enums.OnOff.ON,␣
↪mode=enums.PolicerMode.L1, cir=10_000, cbs=1_000, buffer_size=1_000)
await flow.bandwidth_control.shaper.set(on_off=enums.OnOff.ON,␣
↪mode=enums.PolicerMode.L2, cir=10_000, cbs=1_000, buffer_size=1_000)

resp = await flow.bandwidth_control.shaper.get()
resp.on_off
resp.mode
resp.cir
resp.cbs
resp.buffer_size
```

Statistics

Statistics for Chimera ports.

Clear Counter

Clear all the impairment (duplicate, drop, mis-ordered, corrupted, latency and jitter) statistics for a Chimera port and flows on the port. The byte and packet counts will restart at zero.

Corresponding CLI command: `PE_CLEAR`

```
await port.emulation.clear.set()
```

Corruption

Obtains statistics concerning all the packets corrupted on between this receive port and its partner TX port.

Corresponding CLI command: `PE_CORTOTAL`

```
port_corrupted = await port.emulation.statistics.corrupted.get()
port_corrupted.fcs_corrupted_pkt_count
port_corrupted.fcs_corrupted_pkt_ratio
port_corrupted.ip_corrupted_pkt_count
port_corrupted.ip_corrupted_pkt_ratio
port_corrupted.tcp_corrupted_pkt_count
port_corrupted.tcp_corrupted_pkt_ratio
port_corrupted.total_corrupted_pkt_count
port_corrupted.total_corrupted_pkt_ratio
port_corrupted.udp_corrupted_pkt_count
port_corrupted.udp_corrupted_pkt_ratio
```

Drop Counter

Obtains statistics concerning all the packets dropped between this receive port and its partner TX port.

Corresponding CLI command: `PE_DROPTOTAL`

```
port_drop = await port.emulation.statistics.drop.get()
port_drop.pkt_drop_count_total
port_drop.pkt_drop_count_programmed
port_drop.pkt_drop_count_bandwidth
port_drop.pkt_drop_count_other
port_drop.pkt_drop_ratio_total
port_drop.pkt_drop_ratio_programmed
port_drop.pkt_drop_ratio_bandwidth
port_drop.pkt_drop_ratio_other
```

Duplication Counter

Obtains statistics concerning all the packets duplicated between this receive port and its partner TX port.

Corresponding CLI command: `PE_DUPTOTAL`

```
port_duplicated = await port.emulation.statistics.duplicated.get()
port_duplicated.pkt_count
port_duplicated.ratio
```

Jittered Counter

Obtains statistics concerning all the packets jittered between this receive port and its partner TX port.

Corresponding CLI command: `PE_JITTERTOTAL`

```
port_jittered = await port.emulation.statistics.jittered.get()
port_jittered.pkt_count
port_jittered.ratio
```

Delay Counter

Obtains statistics concerning all the packets delayed this receive port and its partner TX port.

Corresponding CLI command: `PE_LATENCYTOTAL`

```
port_delayed = await port.emulation.statistics.latency.get()
port_delayed.pkt_count
port_delayed.ratio
```

Misordering Counter

Obtains statistics concerning all the packets mis-ordered between this receive port and its partner TX port.

Corresponding CLI command: `PE_MISTOTAL`

```
port_misordered = await port.emulation.statistics.mis_ordered.get()
port_misordered.pkt_count
port_misordered.ratio
```

L47

Note: Applicable to Vulcan port only.

Port Address

ARP Configuration

```
await port.arp_config.set()
await port.arp_config.get()
```

NDP Configuration

```
await port.ndp_config.set()
await port.ndp_config.get()
```

Capabilities & Aptitudes

```
await port.aptitudes.get()
await port.capabilities.get()
```

Capture

Start

```
await port.capture.start.set_on()
await port.capture.start.set_off()
await port.capture.start.get()
```

Read PCAP

```
await port.capture.get_first_frame.get()
await port.capture.get_next_frame.get()
```

Clear All

```
await port.clear.set()
```

Identification

Firmware Version

```
await port.nic_firmware_version.get()
```

NIC Name

```
await port.nic_name.get()
```

Status

```
await port.last_state_status.get()
```

License Info

```
await port.license_info.get()
```

Max Packet Rate

```
await port.max_packet_rate.set_automatic()
await port.max_packet_rate.set_manual()
await port.max_packet_rate.get()
```

Counter

Note: Applicable to Vulcan port only.

Port Counters

Note: Applicable to Vulcan port only.

ARP

Note: Applicable to Vulcan port only.

Total

```
await port.counters.arp.total.get()
```

TX

```
await port.counters.arp.tx.get()
```

RX

```
await port.counters.arp.rx.get()
```

Clear Counters

Note: Applicable to Vulcan port only.

```
await port.counters.clear.set()
```

Ethernet

Note: Applicable to Vulcan port only.

Total

```
await port.counters.eth.total.get()
```

TX

```
await port.counters.eth.tx.get()
```

RX

```
await port.counters.eth.rx.get()
```

ICMP

Note: Applicable to Vulcan port only.

Total

```
await port.counters.icmp.total.get()
```

TX

```
await port.counters.icmp.tx.get()
```

RX

```
await port.counters.icmp.rx.get()
```

IPv4

Total

```
await port.counters.ipv4.total.get()
```

TX

```
await port.counters.ipv4.tx.get()
```

RX

```
await port.counters.ipv4.rx.get()
```

IPv6

Note: Applicable to Vulcan port only.

Total

```
await port.counters.ipv6.total.get()
```

TX

```
await port.counters.ipv6.tx.get()
```

RX

```
await port.counters.ipv6.rx.get()
```

NDP

Note: Applicable to Vulcan port only.

Total

```
await port.counters.ndp.total.get()
```

TX

```
await port.counters.ndp.tx.get()
```

RX

```
await port.counters.ndp.rx.get()
```

TCP

Note: Applicable to Vulcan port only.

Total

```
await port.counters.tcp.total.get()
```

TX

```
await port.counters.tcp.tx.get()
```

RX

```
await port.counters.tcp.rx.get()
```

All

Note: Applicable to Vulcan port only.

Total

```
await port.counters.total.get()
```

TX

```
await port.counters.total_tx.get()
```

RX

```
await port.counters.total_rx.get()
```

UDP

Note: Applicable to Vulcan port only.

Total

```
await port.counters.udp.total.get()
```

TX

```
await port.counters.udp.tx.get()
```

RX

```
await port.counters.udp.rx.get()
```

Connection Group Counters

Note: Applicable to Vulcan port only.

Replay

Note: Applicable to Vulcan port only.

```
await cg.replay.counters.replay.get()
```

TCP

Note: Applicable to Vulcan port only.

Error

```
await cg.tcp.counters.error.get()
```

Packet

```
await cg.tcp.counters.packet.tx.get()
await cg.tcp.counters.packet.rx.get()
```

Payload

```
await cg.tcp.counters.payload.rx.get()
await cg.tcp.counters.payload.tx.get()
```

Retransmission

```
await cg.tcp.counters.retransmission.get()
```

State

```
await cg.tcp.counters.state.current.get()
await cg.tcp.counters.state.rate.get()
await cg.tcp.counters.state.total.get()
```

Histogram

```
await cg.tcp.histogram.connection.close_times.get()
await cg.tcp.histogram.connection.establish_times.get()
await cg.tcp.histogram.rx.good_bytes.get()
await cg.tcp.histogram.rx.total_bytes.get()
await cg.tcp.histogram.tx.good_bytes.get()
await cg.tcp.histogram.tx.total_bytes.get()
```

Clear Port Test Statistics

```
await cg.tcp.clear_post_test_statistics.set()
```

TLS

Note: Applicable to Vulcan port only.

Alerts

```
await cg.tls.counters.alert.fatal.get()

await cg.tls.counters.alert.warning.get()
```

Payload

```
await cg.tls.counters.payload.tx.get()
await cg.tls.counters.payload.rx.get()
```

State

```
await cg.tls.counters.state.current.get()
await cg.tls.counters.state.rate.get()
await cg.tls.counters.state.total.get()
```

Histogram

```
await cg.tls.histogram.handshake.get()
await cg.tls.histogram.payload.rx_bytes.get()
await cg.tls.histogram.payload.tx_bytes.get()
```

Transaction

Note: Applicable to Vulcan port only.

```
await cg.raw.transaction_counter.transaction.get()
```

UDP

Note: Applicable to Vulcan port only.

Packet

```
await cg.udp.counters.packet.rx.get()
await cg.udp.counters.packet.tx.get()
```

Payload

```
await cg.udp.counters.payload.rx.get()
await cg.udp.counters.payload.tx.get()
```

State

```
await cg.udp.counters.state.current.get()
await cg.udp.counters.state.rate.get()
await cg.udp.counters.state.total.get()
```

Histogram

```
await cg.udp.histogram.rx_bytes.get()
await cg.udp.histogram.tx_bytes.get()
```

User

Note: Applicable to Vulcan port only.

```
await cg.user_state.current.get()
await cg.user_state.rate.get()
await cg.user_state.total.get()
```

Connection Group

Note: Applicable to Vulcan port only.

Clear Counters

Note: Applicable to Vulcan port only.

```
await cg.counters.clear.set()
```

Create, Obtain, Remove

Note: Applicable to Vulcan port only.

Create and Obtain

Create a connection group on the Vulcan port, and obtain the connection group object. The connection group index is automatically assigned by the port.

```
cg = await port.connection_groups.create()
```

Obtain One

Obtain an existing connection group on the port with an explicit connection group index.

```
cg = port.connection_groups.obtain(cg_idx)
```

Obtain Multiple

Obtain multiple existing connection groups on the port with explicit connection group indices.

```
cg_list = port.connection_groups.obtain_multiple(*cg_idx_list)
```

Remove

Remove a connection group on the port with an explicit connection group index.

```
await port.connection_groups.remove(cg_idx)
```

Description

Note: Applicable to Vulcan port only.

```
await cg.comment.set()
await cg.comment.get()
```

Histogram

Note: Applicable to Vulcan port only.

Configuration

```
await cg.histogram.config.transaction.set()
await cg.histogram.config.transaction.get()

await cg.histogram.config.payload.set()
await cg.histogram.config.payload.get()

await cg.histogram.config.time.set()
await cg.histogram.config.time.get()
```

Recalculate

```
await cg.histogram.recalculates.transaction.set()
await cg.histogram.recalculates.transaction.get()

await cg.histogram.recalculates.payload.set()

await cg.histogram.recalculates.time.set()
```

Transaction

```
await cg.histogram.transaction.get()
```

L2

Note: Applicable to Vulcan port only.

ARP Resolution

```
await cg.l2.address_resolve.set_no()
await cg.l2.address_resolve.set_yes()
await cg.l2.address_resolve.get()
```

Use Gateway MAC as DMAC

```
await cg.l2.gateway.use.set_yes()
await cg.l2.gateway.use.set_no()
await cg.l2.gateway.use.get()
```

Gateway Config

```
await cg.l2.gateway.ipv4.set()
await cg.l2.gateway.ipv4.get()

await cg.l2.gateway.ipv6.set()
await cg.l2.gateway.ipv6.get()
```

MAC Address

```
await cg.l2.mac.client.set()
await cg.l2.mac.client.get()

await cg.l2.mac.server.set()
await cg.l2.mac.server.get()
```

VLAN Settings

```
await cg.l2.vlan.enable.set_on()
await cg.l2.vlan.enable.set_off()
await cg.l2.vlan.enable.get()

await cg.l2.vlan.tci.set()
await cg.l2.vlan.tci.get()
```

L3

Note: Applicable to Vulcan port only.

IP Version

```
await cg.l3.ip_version.set_ipv4()
await cg.l3.ip_version.set_ipv6()
await cg.l3.ip_version.get()
```

IPv4 Range

```
await cg.l3.ipv4.client_range.set()
await cg.l3.ipv4.client_range.get()

await cg.l3.ipv4.server_range.set()
await cg.l3.ipv4.server_range.get()
```

IPv4 DiffServ Config

```
await cg.l3.diffserv.mask.set()
await cg.l3.diffserv.mask.get()

await cg.l3.diffserv.range_limits.set()
await cg.l3.diffserv.range_limits.get()

await cg.l3.diffserv.step.set()
await cg.l3.diffserv.step.get()

await cg.l3.diffserv.type.set()
```

(continues on next page)

(continued from previous page)

```
await cg.l3.diffserv.type.get()

await cg.l3.diffserv.value.set()
await cg.l3.diffserv.value.get()
```

IPv6 Range

```
await cg.l3.ipv6.client_range.set()
await cg.l3.ipv6.client_range.get()

await cg.l3.ipv6.server_range.set()
await cg.l3.ipv6.server_range.get()
```

IPv6 Flow Label Config

```
await cg.l3.ipv6.flow_label.set()
await cg.l3.ipv6.flow_label.get()

await cg.l3.ipv6.traffic_class.set()
await cg.l3.ipv6.traffic_class.get()
```

NAT

```
await cg.l3.nat.set_on()
await cg.l3.nat.set_off()
await cg.l3.nat.get()
```

Load Profile

Note: Applicable to Vulcan port only.

Shape

```
await cg.load_profile.shape.set()
await cg.load_profile.shape.get()
```

Time Scale

```
await cg.load_profile.time_scale.set_msecs()
await cg.load_profile.time_scale.set_seconds()
await cg.load_profile.time_scale.set_minutes()
await cg.load_profile.time_scale.set_hours()
await cg.load_profile.time_scale.get()
```

Role

Note: Applicable to Vulcan port only.

```
await cg.role.set_client()
await cg.role.set_server()
await cg.role.get()
```

Status

Note: Applicable to Vulcan port only.

```
await cg.status.set_on()
await cg.status.set_off()
await cg.status.get()
```

TLS

Note: Applicable to Vulcan port only.

Enable

```
await cg.tls.enable.set_yes()
await cg.tls.enable.set_no()
await cg.tls.enable.get()
```

Cipher Suites

```
await cg.tls.cipher_suites.set()
await cg.tls.cipher_suites.get()
```

Close Notify

```
await cg.tls.close_notify.set_yes()
await cg.tls.close_notify.set_no()
await cg.tls.close_notify.get()
```

Certificate and Key

```
await cg.tls.file.certificate_path.set()
await cg.tls.file.dhparams_path.set()
await cg.tls.file.private_key_path.set()
```

Max Record Size

```
await cg.tls.max_record_size.set()
await cg.tls.max_record_size.get()
```

Protocol Version

```
await cg.tls.protocol.version.set_sslv3()
await cg.tls.protocol.version.set_tls10()
await cg.tls.protocol.version.set_tls11()
await cg.tls.protocol.version.set_tls12()
await cg.tls.protocol.version.get()
```

Minimum Required Version

```
await cg.tls.protocol.min_required_version.get()
```

Server Name

```
await cg.tls.server_name.set()
await cg.tls.server_name.get()
```

L4

Note: Applicable to Vulcan port only.

L4 Protocol

```
await cg.layer4_protocol.set_tcp()
await cg.layer4_protocol.set_udp()
await cg.layer4_protocol.get()
```

TCP

Note: Applicable to Vulcan port only.

ACK Configuration

```
await cg.tcp.ack.duplicate_thresholds.set()
await cg.tcp.ack.duplicate_thresholds.get()

await cg.tcp.ack.frequency.set()
await cg.tcp.ack.frequency.get()

await cg.tcp.ack.timeout.set()
await cg.tcp.ack.timeout.get()
```

Congestion Control

```
await cg.tcp.cwnd.congestion_mode.set_none()
await cg.tcp.cwnd.congestion_mode.set_new_reno()
await cg.tcp.cwnd.congestion_mode.set_reno()
await cg.tcp.cwnd.congestion_mode.get()
```

Initial CWND

```
await cg.tcp.cwnd.icwnd_calc_method.set_fixed_factor()
await cg.tcp.cwnd.icwnd_calc_method.set_rfc2581()
await cg.tcp.cwnd.icwnd_calc_method.set_rfc5681()
await cg.tcp.cwnd.icwnd_calc_method.get()
```

Initial Slow Slart

```
await cg.tcp.iss_treshold.set_automatic()
await cg.tcp.iss_treshold.set_manual()
await cg.tcp.iss_treshold.get()
```

Max Segment Size - Fixed

```
await cg.tcp.mss.fixed_value.set()
await cg.tcp.mss.fixed_value.get()
```

Max Segment Size - Varying

```
await cg.tcp.mss.range_limits.set()
await cg.tcp.mss.range_limits.get()

await cg.tcp.mss.type.set_fixed()
await cg.tcp.mss.type.set_increment()
await cg.tcp.mss.type.set_random()
await cg.tcp.mss.type.get()
```

RWND Scaling

```
await cg.tcp.rwnd.scaling.set_yes()
await cg.tcp.rwnd.scaling.set_no()
await cg.tcp.rwnd.scaling.get()
```

RWND Size

```
await cg.tcp.rwnd.size.set()
await cg.tcp.rwnd.size.get()
```

Retransmission Timeout Prolonged Mode

```
await cg.tcp.rto.prolonged_mode.set_disable()
await cg.tcp.rto.prolonged_mode.set_enable()
await cg.tcp.rto.prolonged_mode.get()
```

Retransmission Timeout Range

```
await cg.tcp.rto.range_limits.set()
await cg.tcp.rto.range_limits.get()
```

SYN Retransmission Timeout

```
await cg.tcp.rto.syn_value.set()
await cg.tcp.rto.syn_value.get()
```

UDP

Note: Applicable to Vulcan port only.

Packet Size Range

```
await cg.udp.packet_size.range_limits.set()
await cg.udp.packet_size.range_limits.get()
```

Packet Size Type

```
await cg.udp.packet_size.type.set_fixed()
await cg.udp.packet_size.type.set_increment()
await cg.udp.packet_size.type.set_random()
await cg.udp.packet_size.type.get()
```

Packet Size Value

```
await cg.udp.packet_size.value.set()
await cg.udp.packet_size.value.get()
```

Test Application

Note: Applicable to Vulcan port only.

Type

```
await cg.test_application.set_none()
await cg.test_application.set_raw()
await cg.test_application.set_replay()
await cg.test_application.get()
```

Raw

Note: Applicable to Vulcan port only.

Test Scenario

```
await cg.raw.test_scenario.set_echo()
await cg.raw.test_scenario.set_download()
await cg.raw.test_scenario.set_upload()
await cg.raw.test_scenario.set_both()
await cg.raw.test_scenario.get()
```

Utilization

```
await cg.raw.utilization.set()
await cg.raw.utilization.get()
```

Transmission Config

```
await cg.raw.tx.during_ramp.set()
await cg.raw.tx.during_ramp.get()

await cg.raw.tx.time_offset.set()
await cg.raw.tx.time_offset.get()
```

Burst Config

```
await cg.raw.bursty.transmission.set_on()
await cg.raw.bursty.transmission.set_off()
await cg.raw.bursty.transmission.get()

await cg.raw.bursty.config.set()
await cg.raw.bursty.config.get()
```

Connection Close

```
await cg.raw.connection.close_condition.set_none()
await cg.raw.connection.close_condition.set_client()
await cg.raw.connection.close_condition.set_server()
await cg.raw.connection.close_condition.get()
```

Connection Incarnation

```
await cg.raw.connection.incarnation.set_once()
await cg.raw.connection.incarnation.set_immortal()
await cg.raw.connection.incarnation.set_reincarnate()
await cg.raw.connection.incarnation.get()
```

Connection Lifetime

```
await cg.raw.connection.lifetime.set_msecs()
await cg.raw.connection.lifetime.set_seconds()
await cg.raw.connection.lifetime.set_minutes()
await cg.raw.connection.lifetime.set_hours()
await cg.raw.connection.lifetime.get()
```

Connection Repetition

```
await cg.raw.connection.repetitions.set_finite()
await cg.raw.connection.repetitions.set_infinite()
await cg.raw.connection.repetitions.get()
```

Download Request Content

```
await cg.raw.download_request.content.set()
await cg.raw.download_request.content.get()
```

Download Request Server Wait

```
await cg.raw.download_request.server_must_wait.set_yes()
await cg.raw.download_request.server_must_wait.set_no()
await cg.raw.download_request.server_must_wait.get()
```

Download Request Transaction Limit

```
await cg.raw.download_request.transactions_number.set_finite()
await cg.raw.download_request.transactions_number.set_infinite()
await cg.raw.download_request.transactions_number.get()
```

Payload Type

```
await cg.raw.payload.type.set_fixed()
await cg.raw.payload.type.set_increment()
await cg.raw.payload.type.set_longrandom()
await cg.raw.payload.type.set_random()
await cg.raw.payload.type.get()
```

Payload Content

```
await cg.raw.payload.content.set()
await cg.raw.payload.content.get()
```

Payload Repeat Length

```
await cg.raw.payload.repeat_length.set()
await cg.raw.payload.repeat_length.get()
```

Payload Total Length

```
await cg.raw.payload.total_length.set_finite()
await cg.raw.payload.total_length.set_infinite()
await cg.raw.payload.total_length.get()
```

Payload RX Length

```
await cg.raw.payload.rx_length.set_infinite()
await cg.raw.payload.rx_length.set_finite()
await cg.raw.payload.rx_length.get()
```

Replay

Note: Applicable to Vulcan port only.

Utilization

```
await cg.replay.utilization.set()
await cg.replay.utilization.get()
```

Replay File

```
await cg.replay.files.indices.get()
await cg.replay.files.clear_index(replay_file_idx)
await cg.replay.files.name(replay_file_idx)
```

User Incarnation

```
await cg.replay.user.incarnation.set_once()
await cg.replay.user.incarnation.set_immortal()
await cg.replay.user.incarnation.set_reincarnate()
await cg.replay.user.incarnation.get()
```

User Repetition

```
await cg.replay.user.repetitions.set_finite()
await cg.replay.user.repetitions.set_infinite()
await cg.replay.user.repetitions.get()
```

Connection Incarnation

```
await cg.raw.connection.incarnation.set_once()
await cg.raw.connection.incarnation.set_immortal()
await cg.raw.connection.incarnation.set_reincarnate()
await cg.raw.connection.incarnation.get()
```

Connection Lifetime

```
await cg.raw.connection.lifetime.set_msecs()
await cg.raw.connection.lifetime.set_seconds()
await cg.raw.connection.lifetime.set_minutes()
await cg.raw.connection.lifetime.set_hours()
await cg.raw.connection.lifetime.get()
```

Connection Repetition

```
await cg.raw.connection.repetitions.set_finite()
await cg.raw.connection.repetitions.set_infinite()
await cg.raw.connection.repetitions.get()
```

Download Request Content

```
await cg.raw.download_request.content.set()
await cg.raw.download_request.content.get()
```

Download Request Server Wait

```
await cg.raw.download_request.server_must_wait.set_yes()
await cg.raw.download_request.server_must_wait.set_no()
await cg.raw.download_request.server_must_wait.get()
```

Download Request Transaction Limit

```
await cg.raw.download_request.transactions_number.set_finite()
await cg.raw.download_request.transactions_number.set_infinite()
await cg.raw.download_request.transactions_number.get()
```

Payload Type

```
await cg.raw.payload.type.set_fixed()
await cg.raw.payload.type.set_increment()
await cg.raw.payload.type.set_longrandom()
await cg.raw.payload.type.set_random()
await cg.raw.payload.type.get()
```

Payload Content

```
await cg.raw.payload.content.set()
await cg.raw.payload.content.get()
```

Payload Repeat Length

```
await cg.raw.payload.repeat_length.set()
await cg.raw.payload.repeat_length.get()
```

Payload Total Length

```
await cg.raw.payload.total_length.set_finite()
await cg.raw.payload.total_length.set_infinite()
await cg.raw.payload.total_length.get()
```

Payload RX Length

```
await cg.raw.payload.rx_length.set_infinite()
await cg.raw.payload.rx_length.set_finite()
await cg.raw.payload.rx_length.get()
```

8.2.5 Stream

Stream APIs for Valkyrie.

Custom Data Field

Note: Use `await port.payload_mode.set_cdf()` to set the port's payload mode to Custom Data Field.

Field Offset

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 zero (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.

Corresponding CLI command: `PS_CDFOFFSET`

```
# Custom Data Field
# Use await port.payload_mode.set_cdf() to set the port's payload mode␣
↪to Custom Data Field.

# Field Offset
await stream.cdf.offset.set(offset=1)

resp = await stream.cdf.offset.get()
resp.offset
```

Byte Count

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.

Corresponding CLI command: `PS_CDFCOUNT`

```
# Byte Count
await stream.cdf.count.set(cdf_count=1)

resp = await stream.cdf.count.get()
resp.cdf_count
```

Connectivity Check

IPv4 Gateway Address

An IPv4 gateway configuration specified for a stream.

Corresponding CLI command: PS_IPV4GATEWAY

```
# IPv4 Gateway Address
await stream.gateway.ipv4.set(gateway=ipaddress.IPv4Address("10.10.10.1
↪"))

resp = await stream.gateway.ipv4.get()
resp.gateway
```

IPv6 Gateway Address

An IPv6 gateway configuration specified for a stream.

Corresponding CLI command: PS_IPV6GATEWAY

```
# IPv6 Gateway Address
await stream.gateway.ipv6.set(gateway=ipaddress.IPv6Address("::0001"))

resp = await stream.gateway.ipv6.get()
resp.gateway
```

ARP Resolve Peer Address

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.

Note: You need to make sure either the port has a correct gateway or the stream has a correct destination IP address to ARP resolve the MAC address.

Corresponding CLI command: PS_ARPREQUEST

```
# ARP Resolve Peer Address
# You need to make sure either the port has a correct gateway or the␣
↪stream has a correct destination IP address to ARP resolve the MAC␣
```

(continues on next page)

(continued from previous page)

```
↪address.
resp = await stream.request.arp.get()
resp.mac_address
```

PING Check IP Peer

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.

Note: You need to make sure either the port has a correct gateway or the stream has a correct destination IP address to ping.

Corresponding CLI command: `PS_PINGREQUEST`

```
# PING Check IP Peer
# You need to make sure either the port has a correct gateway or the␣
↪stream has a correct destination IP address to ping.
resp = await stream.request.ping.get()
resp.delay
resp.time_to_live
```

Packet Content

Packet Size

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.

Corresponding CLI command: `PS_PACKETLENGTH`

```
# Packet Size
await stream.packet.length.set(length_type=enums.LengthType.FIXED, min_
↪val=64, max_val=64)
await stream.packet.length.set(length_type=enums.LengthType.
↪INCREMENTING, min_val=64, max_val=1500)
await stream.packet.length.set(length_type=enums.LengthType.BUTTERFLY,␣
```

(continues on next page)

(continued from previous page)

```
↪min_val=64, max_val=1500)
await stream.packet.length.set(length_type=enums.LengthType.RANDOM,␣
↪min_val=64, max_val=1500)
await stream.packet.length.set(length_type=enums.LengthType.MIX, min_
↪val=64, max_val=64)

resp = await stream.packet.length.get()
resp.length_type
resp.min_val
resp.max_val
```

Packet Auto Size

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>.

Corresponding CLI command: PS_AUTOADJUST

```
# Packet Auto Size
await stream.packet.auto_adjust.set()
```

Payload Type

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.

Corresponding CLI command: `PS_PAYLOAD`

```
# Payload Type
# Pattern string in hex, min = 1 byte, max = 18 bytes
await stream.payload.content.set(payload_type=enums.PayloadType.
 ↪PATTERN, hex_data=Hex("000102030405060708090A0B0C0D0E0FDEAD"))
await stream.payload.content.set(payload_type=enums.PayloadType.
 ↪PATTERN, hex_data=Hex("F5"))

# Patter string ignored for non-pattern types
await stream.payload.content.set(payload_type=enums.PayloadType.INC16,␣
 ↪hex_data=Hex("F5"))
await stream.payload.content.set_inc_word("00")
await stream.payload.content.set(payload_type=enums.PayloadType.INC8,␣
 ↪hex_data=Hex("F5"))
await stream.payload.content.set_inc_byte("00")
await stream.payload.content.set(payload_type=enums.PayloadType.DEC8,␣
 ↪hex_data=Hex("F5"))
await stream.payload.content.set_dec_byte("00")
await stream.payload.content.set(payload_type=enums.PayloadType.DEC16,␣
 ↪hex_data=Hex("F5"))
await stream.payload.content.set_dec_word("00")
await stream.payload.content.set(payload_type=enums.PayloadType.PRBS,␣
 ↪hex_data=Hex("F5"))
await stream.payload.content.set_prbs("00")
await stream.payload.content.set(payload_type=enums.PayloadType.RANDOM,
 ↪ hex_data=Hex("F5"))
await stream.payload.content.set_random("00")

resp = await stream.payload.content.get()
resp.hex_data
resp.payload_type
```

Extended Payload

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.

Note: Use `await port.payload_mode.set_extpl()` to set the port's payload mode to Extended Payload.

Corresponding CLI command: `PS_EXTPAYLOAD`

```
# Extended Payload
# Use await port.payload_mode.set_extpl() to set the port's payload␣
↪mode to Extended Payload.
await stream.payload.extended.set(hex_data=Hex("00110022FF"))

resp = await stream.payload.extended.get()
resp.hex_data
```

Create, Obtain, Remove

Create and Obtain

Create a stream on the port, and obtain the stream object. The stream index is automatically assigned by the port.

Corresponding CLI command: `PS_CREATE`

```
stream = await port.streams.create()
```

Obtain One

Obtain an existing stream on the port with an explicit stream index.

```
stream = = port.streams.obtain(0)
```

Obtain Multiple

Obtain multiple existing streams on the port with explicit stream indices.

```
stream_list = port.streams.obtain_multiple(*[0,1,2])
```

Remove

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

Corresponding CLI command: PS_DELETE

```
# Remove
# Remove a stream on the port with an explicit stream index.
await port.streams.remove(position_idx=0)
```

Packet Header Definition

Header Protocol Segment

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.

Corresponding CLI command: PS_HEADERPROTOCOL

```
# Header Protocol Segment
await stream.packet.header.protocol.set(segments=[
    enums.ProtocolOption.ETHERNET,
    enums.ProtocolOption.VLAN,
    enums.ProtocolOption.IP,
    enums.ProtocolOption.UDP,
])

# ETHERNET = 1
# """Ethernet II"""
# VLAN = 2
# """VLAN"""
# ARP = 3
# """Address Resolution Protocol"""
# IP = 4
# """IPv4"""
```

(continues on next page)

(continued from previous page)

```
# IPV6 = 5
# """IPv6"""
# UDP = 6
# """User Datagram Protocol (w/o checksum)"""
# TCP = 7
# """Transmission Control Protocol (w/o checksum)"""
# LLC = 8
# """Logic Link Control"""
# SNAP = 9
# """Subnetwork Access Protocol"""
# GTP = 10
# """GPRS Tunnelling Protocol"""
# ICMP = 11
# """Internet Control Message Protocol"""
# RTP = 12
# """Real-time Transport Protocol"""
# RTCP = 13
# """RTP Control Protocol"""
# STP = 14
# """Spanning Tree Protocol"""
# SCTP = 15
# """Stream Control Transmission Protocol"""
# MACCTRL = 16
# """MAC Control"""
# MPLS = 17
# """Multiprotocol Label Switching"""
# PBBTAG = 18
# """Provider Backbone Bridge tag"""
# FCOE = 19
# """Fibre Channel over Ethernet"""
# FC = 20
# """Fibre Channel"""
# FCOETAIL = 21
# """Fibre Channel over Ethernet (tail)"""
# IGMPV3L0 = 22
# """IGMPv3 Membership Query L=0"""
# IGMPV3L1 = 23
# """IGMPv3 Membership Query L=1"""
# UDPCHECK = 24
# """User Datagram Protocol (w/ checksum)"""
# IGMPV2 = 25
# """Internet Group Management Protocol v2"""
# MPLS_TP_OAM = 26
# """MPLS-TP, OAM Header"""
# GRE_NOCHECK = 27
# """Generic Routing Encapsulation (w/o checksum)"""
```

(continues on next page)

(continued from previous page)

```
# GRE_CHECK = 28
# """Generic Routing Encapsulation (w/ checksum)"""
# TCPCHECK = 29
# """Transmission Control Protocol (w/ checksum)"""
# GTPV1L0 = 30
# """GTPv1 (no options), GPRS Tunneling Protocol v1"""
# GTPV1L1 = 31
# """GTPv1 (w/ options), GPRS Tunneling Protocol v1"""
# GTPV2L0 = 32
# """GTPv2 (no options), GPRS Tunneling Protocol v2"""
# GTPV2L1 = 33
# """GTPv2 (w/ options), GPRS Tunneling Protocol v2"""
# IGMPV1 = 34
# """Internet Group Management Protocol v1"""
# PWETHCTRL = 35
# """PW Ethernet Control Word"""
# VXLAN = 36
# """Virtual eXtensible LAN"""
# ETHERNET_8023 = 37
# """Ethernet 802.3"""
# NVGRE = 38
# """Generic Routing Encapsulation (Network Virtualization)"""
# DHCPV4 = 39
# """Dynamic Host Configuration Protocol (IPv4)"""
# GENEVE = 40
# """Generic Network Virtualization Encapsulation"""

resp = await stream.packet.header.protocol.get()
resp.segments
```

Header Value

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.

Corresponding CLI command: `PS_PACKETHEADER`

```
# Header Value
await stream.packet.header.data.set(
    hex_data=Hex(
```

(continues on next page)

(continued from previous page)

```
↪"00000000000004F4BC7FFE90810000000800450000 2A000000007F113BC400000000000000000000000
↪"))

resp = await stream.packet.header.data.get()
resp.hex_data
```

Identification

Description

The description of a stream.

Corresponding CLI command: `PS_COMMENT`

```
# Description
await stream.comment.set(comment="description")

resp = await stream.comment.get()
resp.comment
```

Test Payload ID

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

Corresponding CLI command: `PS_TPLDID`

```
# Test Payload ID
await stream.tpld_id.set(test_payload_identifier=0)

resp = await stream.tpld_id.get()
resp.test_payload_identifier)
```

State

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.

Corresponding CLI command: `PS_ENABLE`

```
# State
await stream.enable.set(state=enums.OnOffWithSuppress.OFF)
await stream.enable.set_off()
await stream.enable.set(state=enums.OnOffWithSuppress.ON)
await stream.enable.set_on()
await stream.enable.set(state=enums.OnOffWithSuppress.SUPPRESS)
await stream.enable.set_suppress()

resp = await stream.enable.get()
resp.state
```

TX Profile

Rate Fraction

The rate of the traffic transmitted for a stream expressed in millionths of the effective rate for the port. The bandwidth consumption includes the inter-frame gap and is independent of the length of the packets generated for the stream. The sum of the bandwidth consumption for all the enabled streams must not exceed the effective rate for the port. Setting this 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.

Corresponding CLI command: `PS_RATEFRACTION`

```
# Rate Fraction
await stream.rate.fraction.set(stream_rate_ppm=1_000_000)

resp = await stream.rate.fraction.get()
resp.stream_rate_ppm
```

Packet Rate

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.

Corresponding CLI command: PS_RATEPPS

```
a# Packet Rate
await stream.rate.pps.set(stream_rate_pps=1_000)

resp = await stream.rate.pps.get()
resp.stream_rate_pps
```

Bit Rate L2

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.

Corresponding CLI command: PS_RATEL2BPS

```
# Bit Rate L2
await stream.rate.l2bps.set(l2_bps=1_000_000)

resp = await stream.rate.l2bps.get()
resp.l2_bps
```

Packet Limit

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.

Corresponding CLI command: PS_RATEPPS

```
# Packet Limit
await stream.packet.limit.set(packet_count=1_000)

resp = await stream.packet.limit.get()
resp.packet_count
```

Burst Size and Density

The burstiness of the traffic transmitted for a stream, expressed in terms of the number of packets in each burst, and how densely they are packed together. The burstiness does not affect the bandwidth consumed by the stream, only the spacing between the packets. A density value of 100 means that the packets are packed tightly together, only spaced by the minimum inter-frame gap. A value of 0 means even, non-bursty, spacing. The exact spacing achieved depends on the other enabled streams of the port.

Corresponding CLI command: PS_BURST

```
# Burst Size and Density
await stream.burst.burstiness.set(size=20, density=80)

resp = await stream.burst.burstiness.get()
resp.size
resp.density
```

Inter Burst/Packet Gap

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).

Corresponding CLI command: PS_BURSTGAP

```
# Inter Burst/Packet Gap
await stream.burst.gap.set(inter_packet_gap=30, inter_burst_gap=30)

resp = await stream.burst.gap.get()
resp.inter_packet_gap
resp.inter_burst_gap
```

Priority Flow

Set and get the Priority Flow Control (PFC) Cos value of a stream.

Corresponding CLI command: `PS_PFCPRIORITY`

```
# Priority Flow
await stream.priority_flow.set(cos=enums.PFCMode.ZERO)
await stream.priority_flow.set(cos=enums.PFCMode.ONE)
await stream.priority_flow.set(cos=enums.PFCMode.TWO)
await stream.priority_flow.set(cos=enums.PFCMode.THREE)
await stream.priority_flow.set(cos=enums.PFCMode.FOUR)
await stream.priority_flow.set(cos=enums.PFCMode.FIVE)
await stream.priority_flow.set(cos=enums.PFCMode.SIX)
await stream.priority_flow.set(cos=enums.PFCMode.SEVEN)
await stream.priority_flow.set(cos=enums.PFCMode.VLAN_PCP)

resp = await stream.priority_flow.get()
resp.cos
```

Modifier

Stream modifier APIs for Valkyrie streams.

Stream 16-bit Modifier

Create

```
# Create
await stream.packet.header.modifiers.configure(number=1)
```

Clear

```
# Clear
await stream.packet.header.modifiers.clear()
```

Obtain

Note: Must create modifiers before obtain.

```
# Obtain
# Must create modifiers before obtain.
modifier = stream.packet.header.modifiers.obtain(idx=0)
```

Range

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... }.

Corresponding CLI command: PS_MODIFIERRANGE

```
# Range
await modifier.range.set(min_val=0, step=10, max_val=9)

resp = await modifier.range.get()
resp.min_val
resp.max_val
resp.step
```

Position, Action, Mask

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.

Corresponding CLI command: PS_MODIFIER

```
# Position, Action, Mask
await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.INC, repetition=1)
await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.DEC, repetition=1)
```

(continues on next page)

(continued from previous page)

```
await modifier.specification.set(position=0, mask=Hex("FFFF0000"),␣
↪action=enums.ModifierAction.RANDOM, repetition=1)

resp = await modifier.specification.get()
resp.action
resp.mask
resp.position
resp.repetition
```

Stream 32-bit Modifier

Create

```
# Create
await stream.packet.header.modifiers_extended.configure(number=1)
```

Clear

```
# Clear
await stream.packet.header.modifiers_extended.clear()
```

Obtain

Note: Must create modifiers before obtain.

```
# Obtain
# Must create modifiers before obtain.
modifier_ext = stream.packet.header.modifiers_extended.obtain(idx=0)
```

Range

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. . . }.

Corresponding CLI command: PS_MODIFIEREXTRANGE

```
# Range
await modifier_ext.range.set(min_val=0, step=1, max_val=100)

resp = await modifier_ext.range.get()
resp.max_val
resp.min_val
resp.step
```

Position, Action, Mask

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.

Corresponding CLI command: PS_MODIFIEREXT

```
# Position, Action, Mask
await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"),␣
↪action=enums.ModifierAction.INC, repetition=1)
await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"),␣
↪action=enums.ModifierAction.DEC, repetition=1)
await modifier_ext.specification.set(position=0, mask=Hex("FFFFFFFF"),␣
↪action=enums.ModifierAction.RANDOM, repetition=1)

resp = await modifier_ext.specification.get()
resp.action
resp.mask
resp.position
resp.repetition
```

Error Handling

Error handling APIs of stream.

Error Injection

Misorder Error

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.

Corresponding CLI command: `PS_INJECTMISERR`

```
# Misorder Error Injection
await stream.inject_err.misorder.set()
```

Payload Integrity Error

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.

Corresponding CLI command: `PS_INJECTPLDERR`

```
# Payload Integrity Error Injection
await stream.inject_err.payload_integrity.set()
```

Sequence Error

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.

Corresponding CLI command: `PS_INJECTSEQERR`

```
# Sequence Error Injection
await stream.inject_err.sequence.set()
```

Test Payload Error

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.

Corresponding CLI command: PS_INJECTTPLDERR

```
# Test Payload Error Injection
await stream.inject_err.test_payload.set()
```

Checksum Error

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.

Corresponding CLI command: PS_INJECTFCSERR

```
# Checksum Error Injection
await stream.inject_err.frame_checksum.set()
```

Insert Frame Checksum

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

Corresponding CLI command: PS_INSERTFCS

```
# Insert Frame Checksum
await stream.insert_packets_checksum.set(on_off=enums.OnOff.ON)
await stream.insert_packets_checksum.set_on()
await stream.insert_packets_checksum.set(on_off=enums.OnOff.OFF)
await stream.insert_packets_checksum.set_off()

resp = await stream.insert_packets_checksum.get()
resp.on_off
```

8.2.6 Exception

`xoa_driver.exceptions` includes exception classes.

exception RepeatedRequestID

Bases: *TransporterException*

exception TransporterException

Bases: *XoaException*

exception WrongModuleError

Bases: Exception

exception WrongTesterError

Bases: Exception

exception WrongTesterPasswordError

Bases: Exception

exception XmpBadCommandError

Bases: *XmpStatusException*

exception XmpBadHeaderError

Bases: *XmpStatusException*

exception XmpBadIndexError

Bases: *XmpStatusException*

exception XmpBadModuleError

Bases: *XmpStatusException*

exception XmpBadParameterError

Bases: *XmpStatusException*

exception XmpBadPortError

Bases: *XmpStatusException*

exception XmpBadSizeError

Bases: *XmpStatusException*

exception XmpBadValueError

Bases: *XmpStatusException*

exception XmpFailedError

Bases: *XmpStatusException*

exception XmpMemoryFailureError

Bases: *XmpStatusException*

exception XmpModuleOperationNotSupportedByChassisError

Bases: *XmpStatusException*

exception XmpNoConnectionError

Bases: *XmpStatusException*

exception XmpNoFreePortLicenseError

Bases: *XmpStatusException*

exception XmpNoLoggedOnError

Bases: *XmpStatusException*

exception XmpNoPeLicenseError

Bases: *XmpStatusException*

exception XmpNotReadableError

Bases: *XmpStatusException*

exception XmpNotReservedError

Bases: *XmpStatusException*

exception XmpNotSupportedError

Bases: *XmpStatusException*

exception XmpNotValidError

Bases: *XmpStatusException*

exception XmpNotWritableError

Bases: *XmpStatusException*

exception XmpPendingError

Bases: *XmpStatusException*

exception XmpStatusException

Bases: Exception

exception XmpXlsDeniedError

Bases: *XmpStatusException*

exception XmpXlsFailedError

Bases: *XmpStatusException*

exception XmpXlsInvalidError

Bases: *XmpStatusException*

exception XoaConnectionError

Bases: *XoaException*

exception XoaConnectionTimeoutError

Bases: *XoaException*

exception XoaException

Bases: Exception

exception XoaLostFuture

Bases: *XoaException*

8.2.7 Enum

`xoa_driver.enums` includes enum classes for Valkyrie, Vulcan, Chimera, ValkyrieVE, and VulcanVE.

8.3 Low-Level API

LL-API contains low-level API classes, giving you the direct control of the tester. The names of the classes are the same as the the CLI commands in *XOA CLI*, making it easy for you to understand the Python API if you are already familiar with XOA CLI.

However, unlike HL-API, LL-API does not provide functionalities such as *auto connection keep-alive* and *auto index management*. This means you need to write more codes to handle those yourself.

The low-level Python APIs are categorized into:

8.3.1 TGA & L1

Chassis

This module contains the **chassis classes** 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.

Module

This module contains the **L23 module classes** 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.

Port

This module contains the **L23 port classes** that deal with basic information about, and configuration of L23 test ports. The L23 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.

Stream

This module contains the **L23 stream classes** deal with configuration of the traffic streams transmitted from a L23 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 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 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.

TX Statistics

This module contains the **L23 port TX statistics classes** 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.

RX Statistics

This module contains the **L23 port RX statistics classes** 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.

High-Speed Port

This module contains the **L23 high-speed port classes** 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.

The high-speed port command names all have the form `PP_<xxx>` and require a module index id and a port index id, and most also require a physical lane index id.

Capture

This module contains the **L23 port capture classes** 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.

Histogram

This module contains the **L23 port histogram classes** 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.

Match Term

This module contains the **L23 port match term classes** 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.

Filter

This module contains the **L23 port filter classes** 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.

Length Term

This module contains the **L23 port length term classes** 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.

Transceiver

This module contains the **L23 port transceiver classes** that deal with access to the register interfaces of the transceiver on a port.

Port Layer-1

This module contains the **Layer-1 classes** that mainly deal auto-negotiation and link training on a Freya port.

8.3.2 L47

Module

This module contains the **L47 module classes**.

Module Packet Engine

This module contains the **L47 module packet engine classes**.

Port

This module contains the **L47 port classes**.

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` are complete. This is where you can read post-test statistics and extract captured packets.

Port Packet Engine

This module contains the **L47 port packet engine classes**.

Connection Group

This module contains the **L47 connection group classes** that deal with configuration of TCP connections and are specific to L47. 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 L47 traffic. A *CG* consists of a number of TCP connections - between one and millions. A CG has a role, which is either client or server. In order to create TCP connections between two ports on a L47 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*.

Note: Connection Group index must start from 0.

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.

A *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. A *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.

Note: Just like client and server range, both the client and server *CGs* must be configured with the *load profile*.

Next the *CG* must be configured with a test application, which defines what kind of traffic is transported in the TCP payload. Currently there are two kinds of test applications:

- `NONE`, which 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`, which 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`, and `both`.

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.

8.3.3 Impairment

Module

This module contains the **impairment module commands**.

Port

This module contains the **impairment port commands**.

The other port commands are the same as *Valkyrie Port Commands*.

Impairment Custom Distribution

This module contains the **impairment custom distribution commands**.

Impairment Distribution

This module contains the **impairment port distribution commands**.

Port Filter

This module contains the **impairment port flow filter commands**.

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.

Port Impairment Statistics

This module contains the **impairment port statistics commands**.

Flow Impairment Statistics

This module contains the **impairment flow statistics commands**.

Flow TX Statistics Classes

This module contains the **impairment flow TX statistics commands**.

Flow RX Statistics Classes

This module contains the **impairment flow RX statistics commands**.

CHAPTER

NINE

GLOSSARY OF TERMS

ANLT
Auto-Negotiation and Link Training.

API
Application Programming Interface.

CG
Connection Group. A Connection Group is a basic building block when creating L47 traffic, and it consists of a configurable number of TCP connections.

CRUD
Create, read, update, and delete operations

DUT
Device Under Test.

HL-API
Xena OpenAutomation High-Level Python API.

HL-FUNC
Xena OpenAutomation High-Level Functions.

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.

Index Manager
An Index Manager manages the subport-level resource indices such as stream indices, filter indices, connection group indices, match term indices, length term indices, etc. It automatically ensures correct and conflict-free index assignment.

LL-API
Xena OpenAutomation Low-Level Python API.

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.

Module Manager
A Module Manager helps you access test modules. There is one Module Manager per

tester.

Module Type
Module Type corresponds to the model of a test module. Modules of different module types have different port counts, port speeds, capabilities, etc. Examples of module types are Loki-100G-5S-1P, Odin-10G-5S-6P-CU.

Port Manager
A Port Managers helps you access test ports. There is one Port Manager per test module.

Port Type
Port Type corresponds to the module that the port belongs to. All ports on the same module have the same port type.

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.

Resource Manager
HL-API provides an easy way to manage subtester test resources, including obtaining test resources and managing indices.

Test Resource
Test chassis, test module, and test port, both hardware and virtual are referred to as test resources. A user must have the ownership of a test resource before be able to perform testing.

TGA
Traffic Generation and Analysis.

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.

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. Read more here.

XOA Core
Xena OpenAutomation Core is an open test suite framework to execute XOA Test Suites as its plugins.

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.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

T

W

X